MacTech Magazine
Volume 20 • Number 05

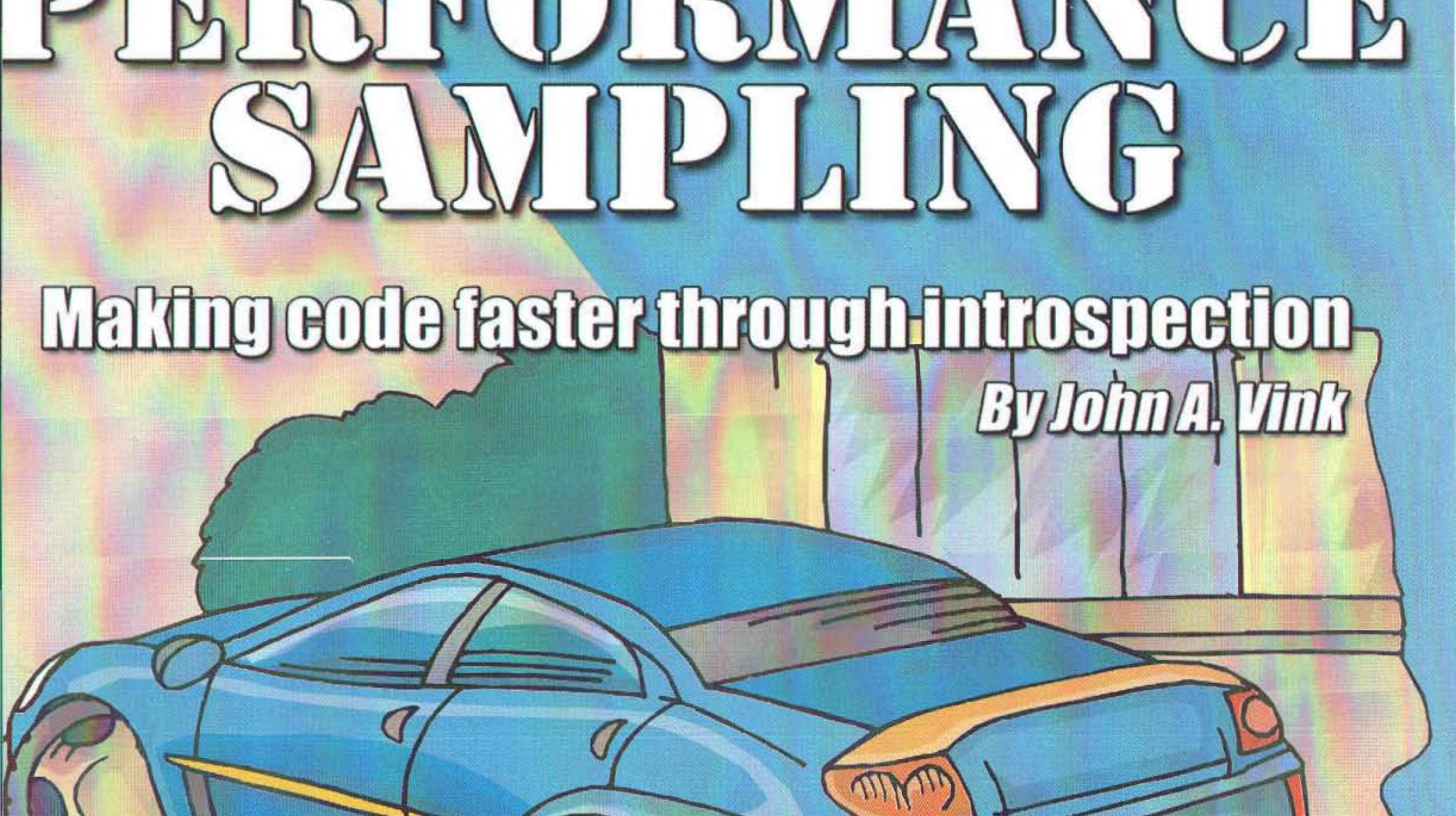
MacTech®

The Journal of Macintosh Technology and Development

PERFORMANCE SAMPLING

Making code faster through introspection

By John A. Vink



Also Inside:

APPLESCRIPT ESSENTIALS: User Interaction Basics
FOCUS REVIEW: Runtime Revolution, Programming for Mere Mortals
HOW TO: "Yoyo" Power Supply Repair
TOOLS: Developing Electronic Hardware on Mac OS X
PROGRAMMING: Programming the BasicStamp
REVIEWS: SoundSticks II by harman multimedia, The Orange Micro iBot Pro

$8.95 US
$12.95 Canada
ISSN 1067-8360
Printed in U.S.A.

Stay In Control Wherever You Go.

For more than a decade, Timbuktu Pro and netOctopus have been the leading remote control, file transfer and systems administration applications for the Mac OS.

Now, both of these indispensable tools are updated to take full advantage of the world's most advanced operating system.

Timbuktu Pro

Whether you're at home or at work, Timbuktu Pro allows you to operate distant computers as if you were sitting in front of them, transfer files or folders quickly and easily, and communicate by instant message, text chat, or voice intercom.
http://www.timbuktupro.com

netOctopus

Intuitive and powerful, netOctopus can manage a network of ten or 10,000 computers. Inventory computers, software and devices on your network; distribute software; configure remote computers; and create custom reports on the fly.
http://www.netoctopus.com

Learn more, try it, or buy it online. Call us at 1-800-485-5741.

timbuktu® · netOctopus®

netopia.

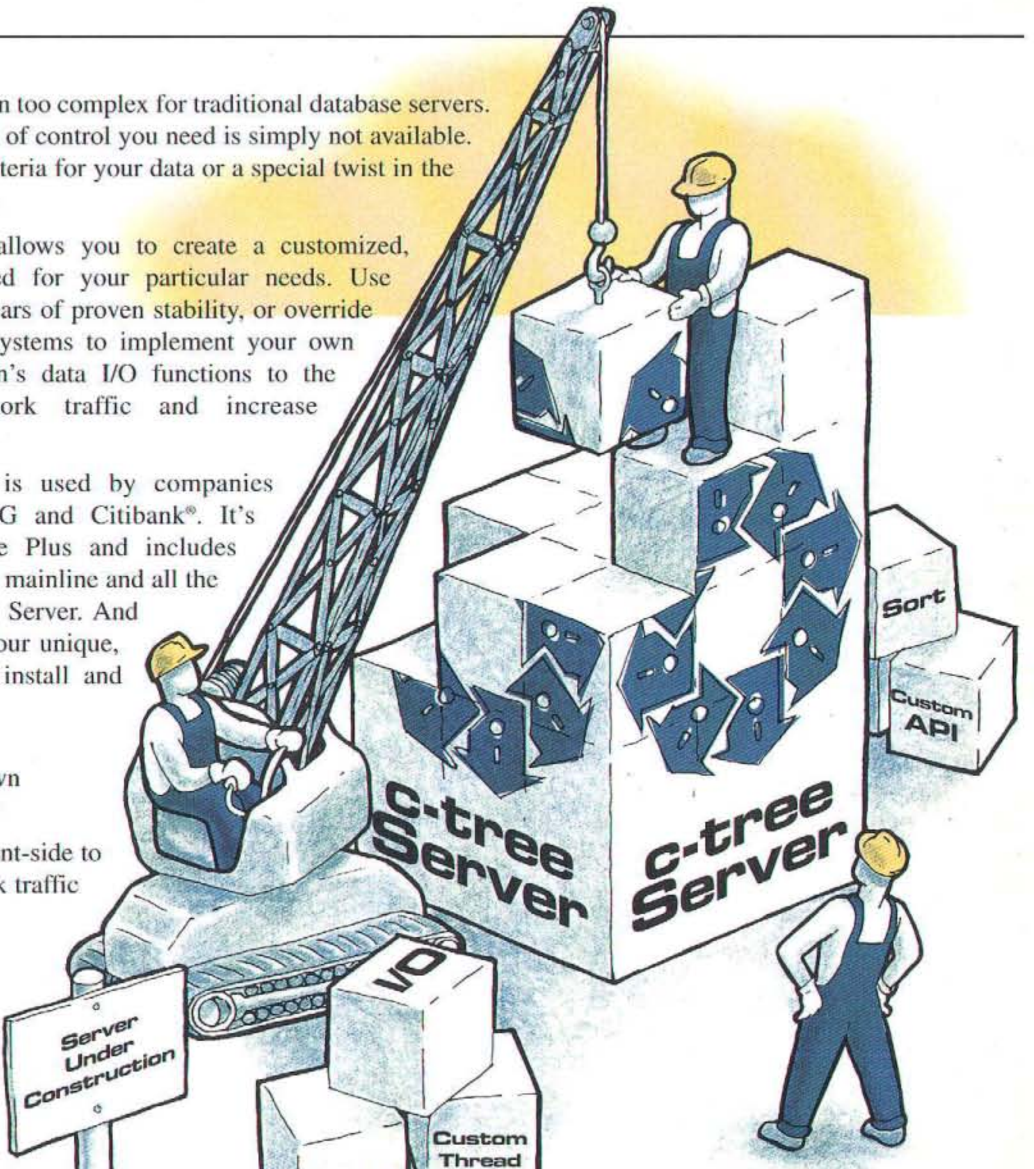
CUSTOMIZE YOUR DATABASE SERVER WITH THE C-TREE® SERVER SDK

Today's database demands are often too complex for traditional database servers. The functionality and precise level of control you need is simply not available. Perhaps you need alternate sort criteria for your data or a special twist in the threading or communication logic.

FairCom's c-tree® Server SDK allows you to create a customized, industrial-strength server designed for your particular needs. Use FairCom's kernel, with over 20 years of proven stability, or override functionality within specific subsystems to implement your own subtleties. Move your application's data I/O functions to the server-side to decrease network traffic and increase performance!

FairCom's c-tree Server SDK is used by companies worldwide such as Software AG and Citibank®. It's integrated seamlessly into c-tree Plus and includes complete source code to the server mainline and all the interface subsystems to the c-tree Server. And best of all, once you've created your unique, customized server, it is easy to install and administer: no DBA required!

- Enhance our server with your own custom server-side functionality
- Move functionality from the client-side to the server-side to reduce network traffic and increase performance
- Modify or replace entire server subsystems
- Complete source for the server mainline, key server subsystems, and client-side
- Flexible OEM licensing



Visit www.faircom.com/ep/mt/sdk today to take control of your server!

FairCom®
www.faircom.com

USA	573.445.6833
EUROPE	+39.035.773.464
JAPAN	+81.59.229.7504
BRAZIL	+55.11.3872.9802

DBMS Since 1979 • 800.234.8180 • info@faircom.com

MacTech®

The Journal of Macintosh Technology & Development

A publication of **XPLAIN** CORPORATION

How To Communicate With Us

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact Developer Depot Customer Service at 877-MACTECH

DEPARTMENTS	E-Mail/URL
Orders, Circulation, & Customer Service	cust_service@mactech.com
Press Releases	press_releases@mactech.com
Ad Sales	ad_sales@mactech.com
Editorial	editorial@mactech.com
Programmer's Challenge	prog_challenge@mactech.com
Online Support	online@mactech.com
Accounting	accounting@mactech.com
Marketing	marketing@mactech.com
General	info@mactech.com
Web Site (articles, info, URLs and more...)	http://www.mactech.com

The MacTech Editorial Staff

Publisher • Neil Ticktin
Editor-in-Chief • Dave Mark
Managing Editor • Jessica Stubblefield

Regular Columnists

Getting Started
by Dave Mark

QuickTime ToolKit
by Tim Monroe

Mac OS X Programming Secrets
by Scott Knaster

Reviews/KoolTools
by Michael R. Harvey

Patch Panel
by John C. Welch

Section 7
by Rich Morin

Untangling the Web
by Kevin Hemenway

John and Pals' Puzzle Page
by John Vink

MacTech's Board of Advisors

Jordan Dea-Mattson, Jim Straus and Jon Wiederspan

MacTech's Contributing Editors

- Michael Brian Bentley
- Vicki Brown
- Marshall Clow
- John. C. Daub
- Tom Djajadiningrat
- Bill Doerrfeld, Blueworld
- Andrew S. Downs
- Richard V. Ford, Packeteer
- Gordon Garb, Sun
- Ilene Hoffman
- Chris Kilbourn, Digital Forest
- Rich Morin
- John O'Fallon, Maxum Development
- Will Porter
- Avi Rappoport, Search Tools Consulting
- Ty Shipman, Kagi
- Chuck Shotton, BIAP Systems
- Cal Simone, Main Event Software
- Steve Sisak, Codewell Corporation
- Chuck Von Rospach, Plaidworks

Xplain Corporation Staff

Chief Executive Officer • Neil Ticktin
President • Andrea J. Sniderman
Controller • Michael Friedman
Production Manager • Jessica Stubblefield
Production/Layout • Darryl Smart
Events Manager • Susan M. Worley
International • Rose Kemps
Network Administrator • David Breffitt
Accounting • Jan Webber, Marcie Moriarty
Customer Relations • Laura Lane, Susan Pomrantz
Shipping/Receiving • Dennis Bower
Board of Advisors • Steven Geller, Blake Park, and Alan Carsrud

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 850-P Hampshire Road, Westlake Village, CA 91361-2800. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Domestic source code disk subscriptions are $77 per year. All international disk subscriptions are $97.00 a year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.

Contents

Volume 20, Issue 5

Foundation's Tool

When I was in college (long ago, boys and girls), one of my all-time favorite classes was Professor Forest's data structures course. We used Knuth's *Sorting and Searching Algorithms* text as our bible and explored all kinds of data structures. My favorite exercise was building a binary tree dictionary implementation using tree balancing techniques to keep the dictionary from leaning too far to one side.

Before we dig into this month's program, I wanted to take a few minutes to explore the thinking behind a simple binary tree dictionary. Even though this doesn't apply directly to our program, I think this is some valuable food for thought, especially if you've never played with binary trees before.

The Binary Tree Dictionary

Figure 1 shows a simple binary tree diagram. The structure that is common to all nodes of the tree features a text string and two pointers, one to the left child and one to the right child. In **Figure 1,** the node holds the text string cat and both child pointers are set to null. Note that there is a root pointer that points to the first node in the tree.

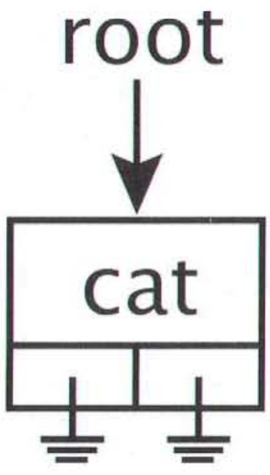


Figure 1. *A binary tree dictionary containing the word cat and two null pointers.*

One way to implement a dictionary is to build a node every time the user enters a new word, then place the node in the tree. A simple method is to insert the first word at the root of the tree, as we did with cat in **Figure 1**. Then, place the next word entered using the left pointer if the word is before cat in the dictionary and using the right pointer if the word is after cat in the dictionary.

As an example, **Figure 2** shows what happens when I insert the word apple into the dictionary structure. Note that the cat node's left child points to the apple node and its right node remains null, while both of the apple nodes are set to null.

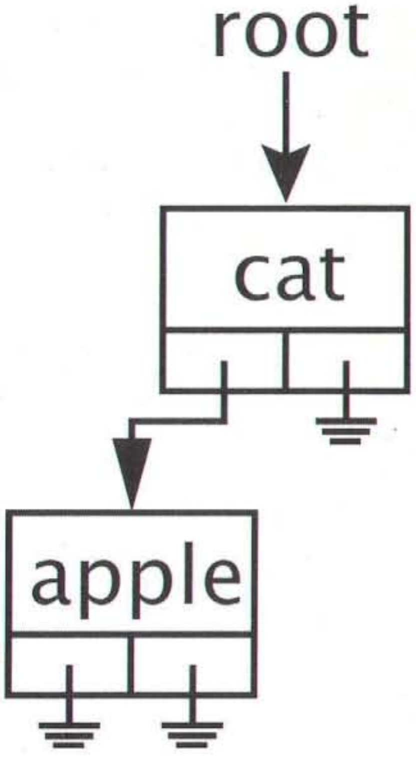


Figure 2. *Adding apple to the dictionary.*

Dave Mark is a long-time Mac developer and author and has written a number of books on Macintosh development, including *Learn C on the Macintosh, Learn C++ on the Macintosh,* and *The Macintosh Programming Primer* series. Dave's been busy lately cooking up his next concoction. Want a peek? http://www.spiderworks.com.

IN THE SOPHOS ZONE, VIRUSES NEVER INTERRUPT YOUR WORKFLOW

Sophos Anti-Virus provides total protection against all known viruses on Mac OS X. Its GUI enables users to perform an immediate scan of any accessible file, folder and volume, while InterCheck technology keeps users safe by intercepting all file accesses and scanning for viruses in the background in real-time. In addition, every license includes 24x7x365 technical support. In the Sophos Zone, viruses don't stand a chance.

For a free, fully supported evaluation, visit: www.sophos.com/av_mac

SOPHOS

engineered for business

www.sophos.com/av_mac
Tel 888-216-6703

Figure 3 shows what happens when you add the word egg to the tree, followed by the word dog and then zebra. Note that dog occurs after cat, but egg is already there. So dog is placed as the left child of egg. When zebra is added to the tree, it goes to the right of cat, then to the right of egg.

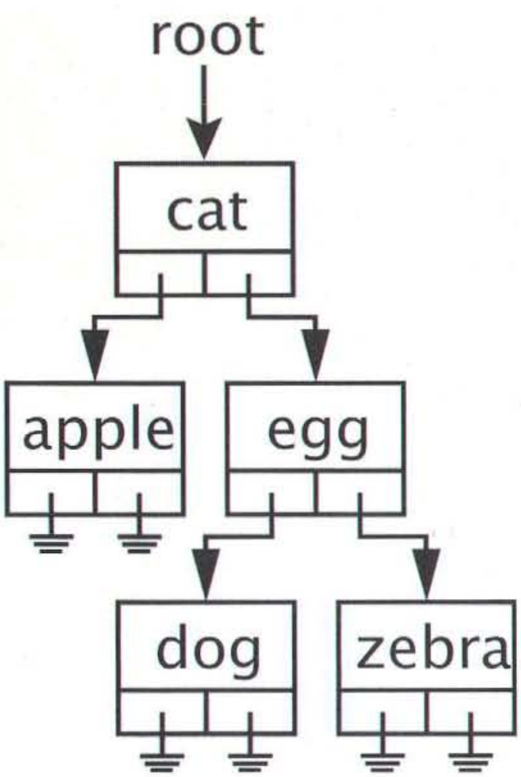


***Figure 3.** More words in the dictionary.*

You can see how you'd add words to the dictionary. To look up a word, you'd start at the root node, then work your way down into the tree. To look up dog, you'd start by comparing dog to cat. Since it is not a match, you'd see that dog comes after cat and follow the right child pointer. When you get to egg, you'd follow the left child pointer (cause dog comes before egg) and you'd find the dog node.

So what happens when you find the dog node? Well, for this to be useful, the dog node would likely have more info in it than just the key value used for lookup. You'd either have an extra pointer to a data block or the data you are looking up would be embedded in the node along with the left and right child pointers.

For example, you might have a pointer to an object containing the definition of the node's word, along with a list of pointers to synonyms of the word. This gives you a structure that serves as both a dictionary and a thesaurus.

One pitfall of this approach is the lack of balance that can occur. For example, imagine what would happen if the very first word entered into the dictionary was *aardvark*. Chances are very good that the left child node would never get used and that the entire dictionary would hang off the right child of the root node. Can you see this?

One solution to this problem is called tree-balancing. After the dictionary is built, it is passed through a tree balancing routine. This function walks the entire tree looking for the key value that belongs exactly in the middle. The tree is then rebuilt with that value as the root node. Again, this is a simple example, but you can see how this would create a more balanced tree. And this turns out to speed up the average search time for the tree. Why? When the tree is more balanced, each level is filled in and the average number or comparisons needed to find a match in the tree goes down. One way to convince yourself of this is to imagine a completely unbalanced, right-leaning tree where every single entry hangs off of a right child. If there are 200 entries in the dictionary, the average search depth will be 100, since the tree will be 200 nodes deep.

In a perfectly balanced tree. You'll have 1 root node, then 2 level 2 nodes, 4 level 3 nodes, 8 level 4 nodes, etc. In that balanced tree, you'll be able to fit 200 nodes quite comfortably in 8 levels of the tree. The worst case search of that tree will require 8 comparisons. *Much* better than the worst case of 200 comparisons in the extremely unbalanced tree.

I love this stuff. Do you find this interesting? If so, let me know and I'd be more than happy to do more in future columns. For now, you might check out these pages:

http://www.semaphorecorp.com/btp/algo.html
http://whatis.techtarget.com/definition/0,289893,sid9_gci508442,00.html

for a nice discussion of b-trees. Note that the second URL has three commas interspersed towards the end.

In the meantime, let's get back to our regularly scheduled Cocoa program.

THE LOTTERY

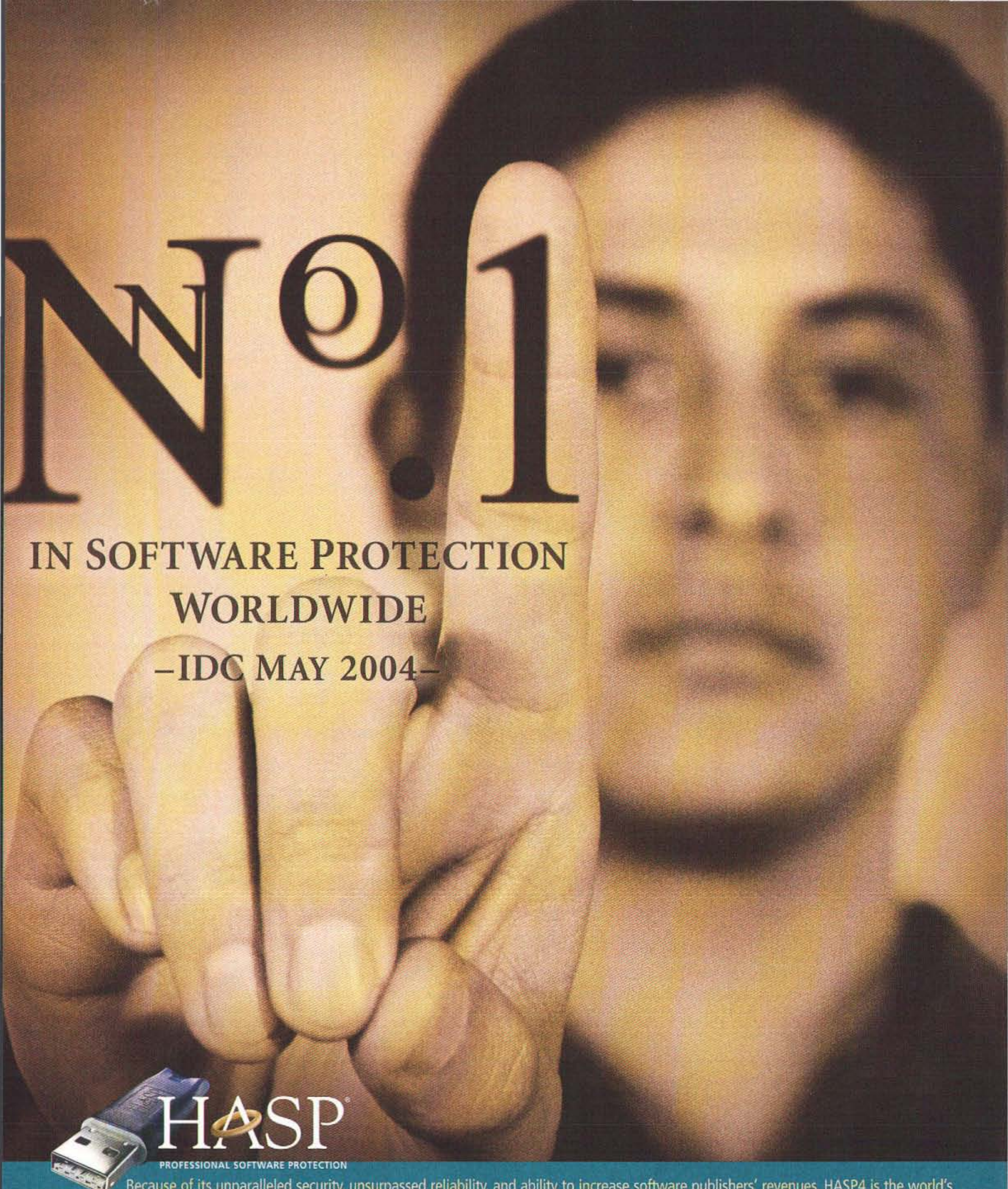
For this month's program, we're going to continue on our journey through Aaron Hillegass' excellent *Cocoa Programming for Mac OS X*. I am fortunate to have in my hot little hands a draft copy of the second edition, which should be hitting bookstores about the time you are reading this.

We'll be building a Foundation Tool project, basically an Objective C program that uses Cocoa but confines its user interface to the command line. The program uses the *NSMutableArray* class to create a list of lottery entries. Aaron claims this will make us fabulously wealthy. We shall see.

Create the Project

Start up Xcode and create a new *Foundation Tool* project. Call it *lottery*. One of the first things you'll notice is Xcode indexing your project, making searches much faster.

In the *Groups & Files* pane, open the *lottery* group and the *Source* subgroup. Now select the file *main.m*. When you click on the file, the contents of main.m should appear in the editing



Because of its unparalleled security, unsurpassed reliability, and ability to increase software publishers' revenues, HASP4 is the world's leading hardware-based software protection. No other company is more committed to the continual advancement of this proven technology for your success. Call 1-800-562-2543 or visit HASP.com/FreeDK for your **FREE HASP4 Developer's Kit** today.

North America: 1-800-562-2543, 847-818-3800 or HASP.us@eAladdin.com **International:** +972-3-636-2222 or HASP.il@eAladdin.com
Germany: HASP.de@eAladdin.com **UK:** HASP.uk@eAladdin.com **France:** HASP.fr@eAladdin.com **Benelux:** HASP.nl@eAladdin.com **Japan:** info@Aladdin.co.jp

Aladdin
SECURING THE GLOBAL VILLAGE
eAladdin.com

pane of the project window. If the editing pane does *not* appear, click on the *Show Editor* icon in the toolbar. It's the fourth icon, just to the right of the stop sign icon.

Here's the default code in main.m:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

  // insert code here...
    NSLog(@"Hello, World!");
    [pool release];
    return 0;
}
```

Replace the existing code with the following:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
  NSMutableArray *array;
  int i;
  NSNumber *newNumber;
  NSNumber *numberToPrint;

  NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

  array = [[NSMutableArray alloc] init];
  for ( i=0; i<10;i++) {
    newNumber = [[NSNumber alloc] initWithInt:(i*3)];
    [array addObject:newNumber];
  }

  for ( i=0; i<10; i++ ) {
    numberToPrint = [array objectAtIndex:i];
    NSLog( @"The number at index %d is %@", i,
        numberToPrint );
  }

  [array release];
  [pool release];
  return 0;
}
```

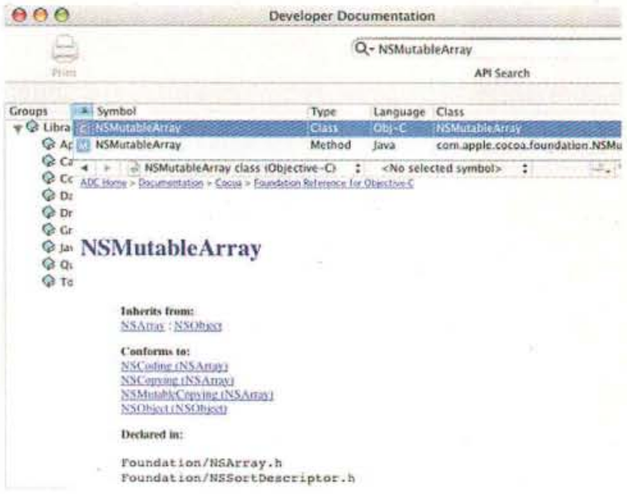
Danger, Will Robinson!!! There is a memory leak in the program you've just entered. If you have a bit of Cocoa experience, see if you can figure out what we did wrong here. Not to worry, though. We'll definitely fix the leak later on in the column.

Build and run the program. Something similar to the following should appear in your Run window:

```
2004-03-29 20:26:52.491 lottery[553] The number at index 0 is 0
2004-03-29 20:26:52.520 lottery[553] The number at index 1 is 3
2004-03-29 20:26:52.532 lottery[553] The number at index 2 is 6
2004-03-29 20:26:52.554 lottery[553] The number at index 3 is 9
2004-03-29 20:26:52.561 lottery[553] The number at index 4 is 12
2004-03-29 20:26:52.567 lottery[553] The number at index 5 is 15
2004-03-29 20:26:52.574 lottery[553] The number at index 6 is 18
2004-03-29 20:26:52.581 lottery[553] The number at index 7 is 21
2004-03-29 20:26:52.589 lottery[553] The number at index 8 is 24
2004-03-29 20:26:52.596 lottery[553] The number at index 9 is 27

lottery has exited with status 0.
```

In a nutshell, this code builds a modifiable (i.e., mutable) array of objects, then stores a pointer to an NSNumber object in each array element.

Whenever I encounter a Cocoa class I haven't seen before, I like to use Xcode to look the class up in the documentation. Hold down the *option* key and double-click on the word *NSMutableArray* in the source code. After a bit of searching, Xcode will bring up the Developer Documentation window (see **Figure 4**) containing a detailed description of the NSMutableArray class. You'll see that it inherits from the NSArray class, that it conforms to a variety of protocols (guarantees to offer a specific set of methods laid out in each protocol declaration), and find out where its headers are declared.



Symbol	Type	Language	Class
NSMutableArray	Class	Obj-C	NSMutableArray
NSMutableArray	Method	Java	com.apple.cocoa.foundation.NSMu

NSMutableArray class (Objective-C) <No selected symbol>

ADC Home > Documentation > Cocoa > Foundation Reference for Objective-C

NSMutableArray

Inherits from:
NSArray : NSObject

Conforms to:
NSCoding (NSArray)
NSCopying (NSArray)
NSMutableCopying (NSArray)
NSObject (NSObject)

Declared in:

```
Foundation/NSArray.h
Foundation/NSSortDescriptor.h
```

Figure 4. *The Developer Documentation window that appears when you OPTION double-click on NSMutableArray.*

Here are a few lines from the doc that are definitely worth their weight in code:

> The NSMutableArray class declares the programmatic interface to objects that manage a modifiable array of objects. This class adds insertion and deletion operations to the basic array-handling behavior inherited from NSArray.
>
> NSMutableArray methods are conceptually based on these primitive methods:
>
> addObject:
>
> insertObject:atIndex:
>
> removeLastObject
>
> removeObjectAtIndex:
>
> replaceObjectAtIndex:withObject:

The other methods in its interface provide convenient ways of inserting an object into a specific slot in the array and removing an object based on its identity or position in the array.

When an object is removed from a mutable array, it receives a release message. If there are no further references to the object, the object is deallocated. Note that if your program keeps a reference to such an object, the reference will become invalid unless you remember to send the object a retain message before it's removed from the array. For example, if anObject isn't retained before removing it from the array, the third statement below could result in a runtime error:

```
id anObject = [[anArray objectAtIndex:0] retain];
[anArray removeObjectAtIndex:0];
[anObject someMessage];
```

Note that the primitive methods listed are links to the detailed description of those methods found later in the same window. Think of NSMutableArray as a linked list of objects where you can add objects to the list at a specific index and replace an object in the list with a different object that you specify.

The term *mutable* means changeable. An NSMutableArray is something you can change. An NSArray is immutable, meaning once you specify it, you are stuck with it. Why would you ever want an NSArray? One classic use is when you want to read in the contents of a file (perhaps a series of records) and store it into a data structure for reference. You don't want to change the data, you just need to access it. NSArray will do the trick.

Let's take a look at the code…

Walking Through the Lottery Code.

Though this source code is quite short, it touches on a number of important concepts. You've already learned a bit about the NSMutableArray. Be sure to read the NSMutableArray doc and look ever the set of methods available to create and modify them.

The source code starts off in typical fashion, with the #import of the Foundation.h header file and the definition of main().

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
```

Next up are declarations of an NSMutableArray object pointer, a counter, and a pair of NSNumber pointers.

```
  NSMutableArray *array;
  int i;
  NSNumber *newNumber;
  NSNumber *numberToPrint;
```

Fetch

Play Fetch with a panther.

X

Fetchsoftworks.com

(Running dog included.)

Any idea what an *NSAutoReleasePool* does? Option-double-click on the class name and Xcode will bring up a doc window that describes the class. Note the link to *Memory Management* toward the middle of the page. Clicking on this link will bring up an intro to memory management as well as an index to a terrific series of articles on Objective C memory management. Take some time to read the intro and, at the very least, scan the other articles to get a sense of the available information. To truly master Cocoa, you must understand Cocoa's unusual memory management techniques.

Here's the short version. Every object has a *retain count.* When an object is created, its retain count is set to 1. If the retain count goes to 0, the object is deallocated. When a chunk of code wants an object to stick around, it sends the object a *retain* message. This increases the retain count by 1. When the chunk of code is done with the object, it sends the object a *release* message. This decrements the retain count by 1. This scheme is well worth understanding and enables many objects to share a common object. When the last object is done using the shared object, the last release message is sent to the shared object and it is deallocated.

The NSAutoReleasePool helps solve a knotty problem common to many frameworks. It is easy to allocate an object at the beginning of a function, then release it at the end of the function. The same object allocs the object and deallocs the object, making it easy to manage. But what if you write a function that creates an object then returns that object to another method? Who deallocates the object?

Java solves this problem by automated garbage collection. Cocoa uses the NSAutoReleasePool. When you send an *autorelease* message to an object, it is added to the autorelease pool. When the pool is deallocated, it sends a release message to all objects in the pool.

We'll talk more about NSAutoreleasePool in future columns. For now, just know that it exists.

```
NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
```

The real key to this column is the NSMutableArray named array. We start by creating array, sending it the alloc message and then the init message. As we've discussed, sending a message to an object is similar to calling an object's method, but without the binding. The actual method called to process the alloc method for the NSMutableArray array is not decided until runtime.

Also worth noting is that messages can be nested. This code sends the alloc message first, then sends the init message. This is a typical way of creating an object.

```
array = [[NSMutableArray alloc] init];
```

Remember, an NSMutableArray is an array of pointers to objects. So if we want to store objects in the array, we need to create them first. Here's a loop that creates a series of 10 NSNumber objects. For each one, we send the message initWithInt: instead of just plain init: when we create the object.

Once the object is created, we add the object to array by sending it the addObject: message. Hmmm. When we add the object to the array, the array no doubt sends a retain message increasing the retain count for each NSNumber to 2. And when we release the array down below, the objects in the array are sent a release message, giving each NSNumber a retain count of 1. But the retain count never gets to 0. Hmmm.

```
for ( i=0; i<10;i++) {
   newNumber = [[NSNumber alloc] initWithInt:(i*3)];
   [array addObject:newNumber];
}
```

What we should have done was add this line of code to the bottom of the for loop, just below the addObject message. Once the NSNumber is added to the array, its retain count is bumped to 2. The release would drop it back down to 1. Then, when array is released and all its NSNumber objects are sent their own release messages, they will be deallocated. See how this works?

```
[newNumber release];
```

Next, we step through the array, using NSLog() to print each of the objects in the array. NSLog() behaves much like printf(), using format descriptors embedded in the string to print the objects that follow the string. While %d should be familiar to you C programmers, %@ is a new beast. Basically, it tells NSLog() to send a *description* message to the object and the object returns a string describing itself. In this case, NSNumber sends a string consisting of its number.

```
for ( i=0; i<10; i++ ) {
   numberToPrint = [array objectAtIndex:i];
   NSLog( @"The number at index %d is %@", i,
        numberToPrint );
}
```

Finally, we release the array and then the autorelease pool.

```
  [array release];
  [pool release];
  return 0;
}
```

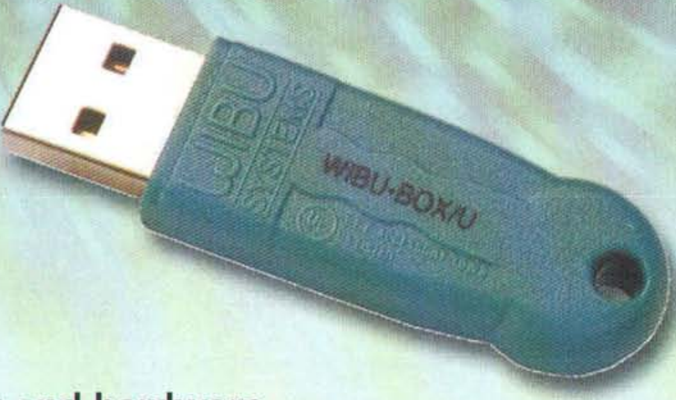
Till Next Month...

Cocoa is an incredibly rich, incredibly powerful framework. There are some things that take getting used to but, once you do, I think you'll find your coding time to be much more productive. I strongly recommend spending some time curled up with your favorite Cocoa book. Then go write some code.

Be sure to check out http://spiderworks.com and I'll see you next month...☺

WIBU-KEY Software Protection

Only hardware can securely protect software



- Common API across all platforms and hardware
- Hardware-based code and data encryption
- Field-upgradable and reusable hardware
- The most cost-effective network licensing solution available
- The only system to employ RID/RED and AXAN security
- Engineered and manufactured to exacting ISO9001 standards

Test the WIBU-KEY Protection Kit
Call 800-986-6578
sales@griftech.com

The Key is in Your Hands!

WIBU SYSTEMS

WIBU-SYSTEMS USA, Inc.
Seattle, WA 98101
Email: info@wibu.com

www.griftech.com
www.wibu.com

Protection Kits also available at:
Belgium wibu@impakt.be, **Czech Republic** info@elbacom-slo.com, **Denmark** lean@danbit.dk, **France** info@neol.fr, **Hungary** info@mrsoft.hu, **Japan** info@suncaria.co.jp, **Jordan, Lebanon** starsoft@cyberia.net.lb, **Korea** dhkimm@wibu.co.kr, **Luxembourg** wibu@impakt.be, **Netherlands** wibu@impakt.be, **Poland** info@elbacom-slo.com, **Portugal** dubit@dubit.pt, **Slovakia** info@elbacom-slo.com, **Slovenia** info@elbacom-slo.com, **Thailand** preecha@dptf.co.th, **United Kingdom** info@codework.com, **USA** sales@griftech.com

By Scott Knaster

Shell Game: Calling Shell Commands from Applications, Part 2

Well, I really hope you saw last month's column, because it was a fun introduction into how to call a shell command from a Cocoa application, which is a neat trick. If you didn't read that column, stick around anyway – we'll recap. This month, we'll extend our knowledge by performing input and output with a shell command from a Cocoa app.

Task Oriented

Last month, we explained how to use Cocoa class NSTask to represent a shell command that we want to execute. We discussed a few methods of NSTask:

setLaunchPath, which specifies the path to the shell command we call from our application.

setArguments, used to pass arguments to the shell command, which you normally do by typing them on the command line itself in Terminal.

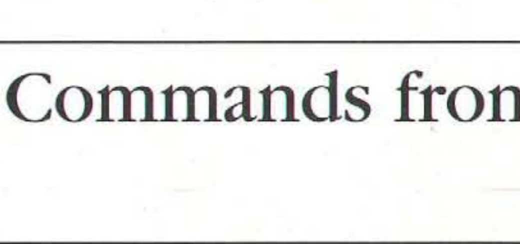
launch, which starts the process running.

These methods get us the basic features we need to run a shell command from a Cocoa app. This time, we're going to take a broader look at what you can do with NSTask. When you create and launch a new task, it gets its own little world to run in. Part of this world includes its own copy of environment variables and its own setting for the current directory. NSTask has methods that let you specify these values. Call setEnvironment to change the process's environment variables to its own custom set, which you pass in as a dictionary. Use setCurrentDirectoryPath to pass a string indicating the directory the task should use as the base for relative paths.

When you call a shell command from Terminal, you can set paths for standard input, standard output, and standard error. These let you choose the source of input for the command, specify where the output should go, and indicate what to do with error messages. NSTask has methods for each of these: setStandardInput, setStandardOutput, and setStandardError. When you call these methods, you pass an object, either an NSPipe or an NSFileHandle, as the input or output. In the next section, we'll discuss what on earth these classes are.

Piece Pipe

NSFileHandle is a class that provides a Cocoa-licious way to exercise control over reading and writing files. NSPipe represents a Unix pipe, a communication channel that reads data from one process and writes it to another. In Cocoa, NSPipe is implemented using two NSFileHandles, one for reading and one for writing. You can use an NSPipe together with an NSTask to send data to a shell command's standard input and then get data from its standard output. As an example, we'll create a application that takes a list of words in an NSTextView and alphabetizes the words by calling the sort shell command. The alphabetized list is then displayed in a another NSTextView. This example is based on code from the course material for the awesome Big Nerd Ranch Cocoa Boot Camp (http://www.bignerdranch.com).



```
Felipe     Bebop
Mateo      Bob
Jesus      Boog
Toot       Felipe
Bob        Hullub
Skip       Jesus
Bebop      Mateo
Boog       Skip
Hullub     Toot
Walleeb    Walleeb
```

Figure 1. *This is what our finished application will look like.*

First, go to Xcode and create a new Cocoa Application project. We'll begin our app with the part that's easy to do and so hard to describe in text: the Interface Builder portion. In IB, in the main window, add two NSTextView objects, with a button between them. When you run this app, you'll type words in the first text view, then click the button, and the words will appear, sorted, in the second text view.

Now let's make the obligatory AppController class and object. Select NSObject on the Classes tab and choose Classes ® Subclass NSObject. Name the new class AppController. Next, open the Info window (Tools ® Show Info), go to the Attributes screen, and add two outlets (inText and outText) and one action (sort). Create the source files for Xcode by choosing Classes ® Create Files for AppController. Finally, add an instance of your AppController class by choosing Classes ® Instantiate AppController.

Now it's time to get our objects to partner up. We want AppController to be connected to both text views, one for input and the other for output. First, Control-drag from the AppController

Scott Knaster writes all sorts of stuff for all kinds of people to read, but nothing is more fun than messing around with Cocoa and OS X. Scott's latest book is *Mac Toys*, from Wiley Publishing. Scott eats a nutritious assortment of fruits and vegetables.

instance to the text view on the left. Double-click inText. Then Control-drag from the AppController to the text view on the right, and double-click outText to connect it. Finally, we need to make the button perform the sort method. Control-drag from the button to AppController, make sure you're on the Target/Action tab of the info window, then double-click sort. Bueno.

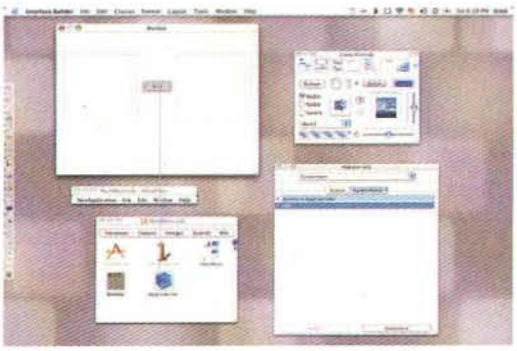


Figure 2. *Lay out the user interface in Interface Builder, as usual.*

Revealing the Code

Now it's time to write the code, so warm up your brain and your typing fingers. All our code will be the implementation for the sort method in AppController.m. First things first:

```
- (IBAction)sort:(id)sender
{
  NSTask *task;
     //Task object that will call "sort"

  NSData *sortResult;
     // Data object for grabbing sorted text

  NSFileHandle *fileToWrite;
     // Handle to standard input for pipe

  NSPipe *inputPipe, *outputPipe;
     //The pipes themselves, for input and output

  NSString *typedText;
     // Holds text typed by the user in the text view

  NSString *sortedText;
     // Will hold sorted text after user clicks button

  task = [[NSTask alloc] init];
  inputPipe = [[NSPipe alloc] init];
  outputPipe = [[NSPipe alloc] init];
     // Don't forget to release these later
```

After declaring and initializing variables, it's time to get the sort command going. We need to create a task and tell it that it's going to execute sort, set up its other values, then send it out into the world, like so:

```
  [task setLaunchPath:@"/usr/bin/sort"];
     // Set the path to the executable. You can find
     // the path for a shell command by using the
     // command "which" in Terminal, e.g. "which sort".

  [task setStandardOutput: outputPipe];
  [task setStandardInput: inputPipe];
     // Set the standard input and standard output
     // for the task. By assigning these to pipes,
     // we can feed input to the task and read its output.

  [task setArguments: [NSArray arrayWithObject:@"-f"]];
     // Set the arguments for sort. We're setting one
     // argument, -f, which specifies a case-insensitive
     // sort (you know, the way humans like it).

  [task launch];
     //The task is all ready to go - start it up!
```

Now that sort is ready to run, it's time for us to supply it with something to sort. We'll take the text the user has typed into the view on the left and ship it off to the process, using a pipe:

```
  fileToWrite = [inputPipe fileHandleForWriting];
     // Get a handle to the pipe that we can use
     // for writing.

  typedText = [inText string];
     // Extract the text out of the
     // first text view (inText).

  [fileToWrite writeData:[
    typedText dataUsingEncoding: NSASCIIStringEncoding]];
     // We've got the text out of the field. Now turn around
     // and write it to the task's standard input as ASCII.
     // Mmm...ASCII.

  [fileToWrite closeFile];
     // Close the pipe when we're done writing.
```

The sort command can now execute, using the text we've written to its standard input. The command proceeds to do its sorting on the input, then writes the sorted result to standard output. We previously hooked up standard output to outputPipe, so now we can deal with the sorted text:

```
  sortResult =
   [[outputPipe fileHandleForReading] readDataToEndOfFile];
     // Grab the sorted text by getting a handle for reading
     // the pipe, then read all the data.

  sortedText = [[NSString alloc] initWithData: sortResult
                      encoding: NSASCIIStringEncoding];
     // Encode the text as ASCII and put the result into
     // sortedText.

  [outText setString: sortedText];
     // Copy the string into the outText view (the one
     // on the right).
```

At this point, we're basically finished – the sorted text appears in the text view on the right. But just like Mom said, we need to clean up before we're completely done.

```
  [sortedText release];
  [task release];
  [inputPipe release];
  [outputPipe release];
```

All About the Process

Now that you've seen how to wrap a shell command in a Cocoa application, you can use shell commands as you need them in your apps. As we saw in this month's installment, you don't have to show any sign of the underlying command if you don't want to – just get control of standard input and standard output, and you can keep the shell command hidden.

For fun projects that use this technique, try writing apps that wrap your own favorite shell commands. Or, for extra credit, you could write a Cocoa app that lets the user type any shell command and its arguments, then executes that command and shows its output. Whatever you do, remember that it's your computer -- you're in command!

QUICKTIME TOOLKIT

by Tim Monroe

Modern Times

Updating the QTShell Application Framework

Introduction

We began this series of articles by developing a simple C-based application called QTShell that runs on both Windows and Macintosh operating systems. QTShell can open and display QuickTime movies, and it supports the standard movie editing operations. Over the past several years, we've gradually tinkered with QTShell to add various capabilities. For instance, in an earlier article ("2001: A Space Odyssey" in *MacTech*, January 2001), we upgraded the Macintosh portions of the code to use the Navigation Services APIs instead of the Standard File Package APIs we used originally (and still use in our Windows code). This was an important step on the road to full Carbonization, which allowed QTShell to run natively on Mac OS X as well as on Mac OS 8 and 9. And in another article ("Event Horizon" in *MacTech*, May 2002), we saw how to switch to the Carbon event model of processing events.

In this article, I want to present several more enhancements to QTShell. The Navigation Services functions that it currently calls, NavGetFile and NavPutFile, are now deprecated; they still work just fine, but they are no longer recommended. By moving to the more modern APIs provided by Navigation Services 3.0, we can pave the way for support for Unicode filenames and for displaying the Save As dialog box as a sheet, as in **Figure 1.**

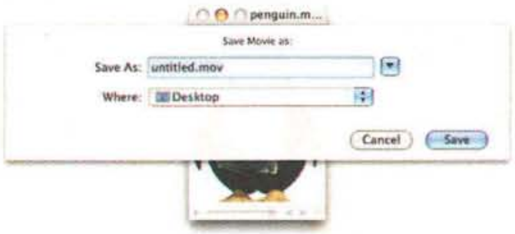


Figure 1: *The Save As sheet*

This is a nicer interface than the dialog box displayed by NavPutFile, which is shown in **Figure 2.**

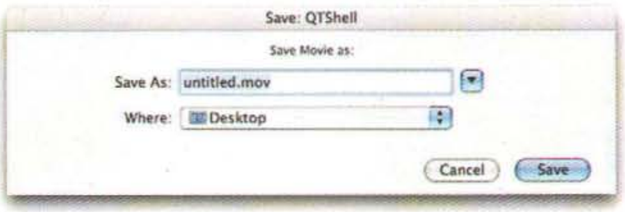


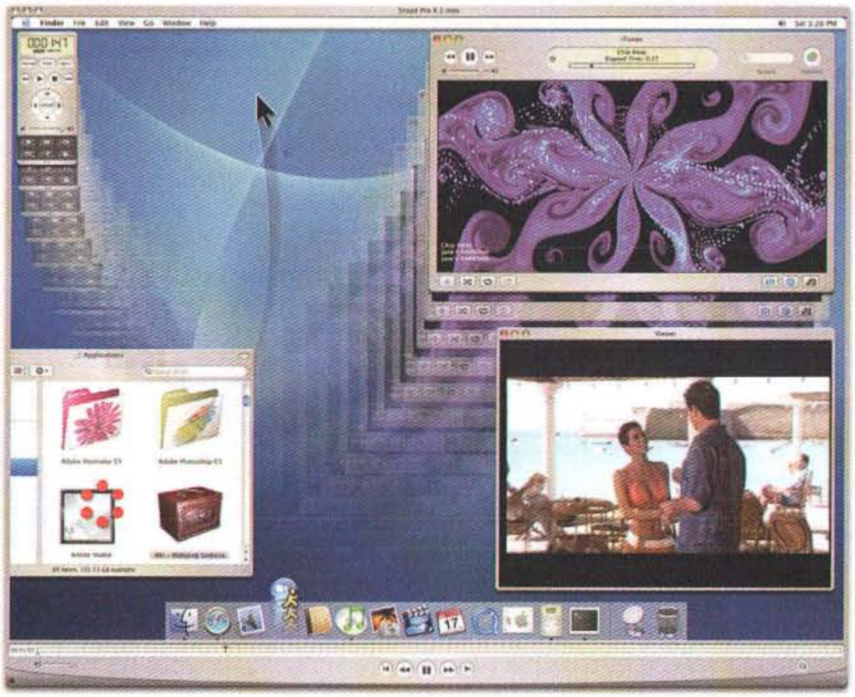
Figure 2: *The Save As dialog box*

I also want to show how to convert QTShell to use the *movie storage* functions introduced in QuickTime 6. A movie storage container is simply any container that can be addressed using a data reference, for instance a file or a block of memory. Currently QTShell works only with files specified using file system specification records (of type FSSpec). In an earlier article ("Somewhere I'll Find You" in *MacTech*, October, 2000), however, we how to open local and remote movie files using NewMovieFromDataRef with file and URL data references. It would be nice to operate on all QuickTime movie data using a single set of APIs, and that's what the movie storage functions provide. Instead of calling OpenMovieFile and specifying a file using an FSSpec, we can call OpenMovieStorage and specify a storage container using a data reference. Then, when it's time to save changes to a movie, we can call UpdateMovieInStorage instead of UpdateMovieResource. And so on. To complement these storage APIs, QuickTime 6.4 introduced a large number of *data reference utilities* that can create data references from data of type FSSpec, CFString, FSRef, CFURL and a handful of other types.

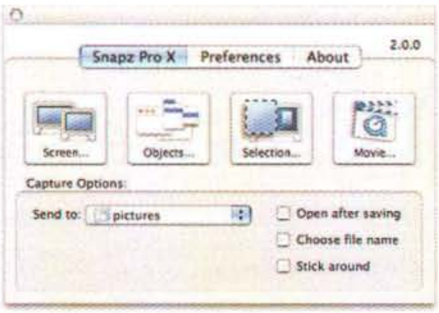
The ultimate goal in moving to the new Navigation Services APIs and the movie storage APIs is to be able to expunge all traces of file system specification records from QTShell. The main problem with FSSpecs is that they cannot represent files with non-ASCII Unicode names or names longer than 63 characters. These other data types — CFString, FSRef, and CFURL — can easily represent Unicode filenames and very long filenames.

Tim Monroe is a member of the QuickTime engineering team at Apple. You can contact him at monroe@mactech.com. The views expressed here are not necessarily shared by his employer.

If a picture is worth a thousand words, imagine how priceless a movie would be...



Snapz Pro X 2.0 allows you to effortlessly record anything on your screen, saving it as a QuickTime® movie that can be emailed, put up on the web, or passed around however you please.



Capture Options:

Send to: pictures

- [ ] Open after saving
- [ ] Choose file name
- [ ] Stick around

Why take a static screenshot when Snapz Pro X 2.0 makes creating a movie just as easy?
Snapz Pro X 2.0 does that, and so much more -- what a difference a version makes!
Download a free demo version from our web site today and see for yourself:

http://www.AmbrosiaSW.com/

Snapz Pro X 2.0

AMBROSIA® SOFTWARE INC

Snapz Pro X

Unfortunately, the complete removal of FSSpec data values from QTShell and all the associated utilities files that our applications depend upon, on both Macintosh and Windows, would require an overhaul that is beyond the scope of this article. But we'll do enough of the groundwork that finally making the jump to an FSSpec-free application will not be too difficult.

File Selection

Let's begin by getting rid of our calls to NavGetFile and NavPutFile. Navigation Services 3.0 and later replace these functions with a handful of functions that allow greater control over the file-selection process. They allow us to display the file-saving dialog box as a sheet (as in **Figure 1**) and they support retrieving information about selected files in the form of an FSRef, which supports Unicode and long filenames.

Choosing a File to Open

Currently QTShell calls the NavGetFile function to display the standard file-opening dialog box, shown in **Figure 3**. NavGetFile handles everything involved in displaying the dialog box and handling user actions in the box. When it exits, it fills out a record of type NavReplyRecord that contains information about the selected file, if any.

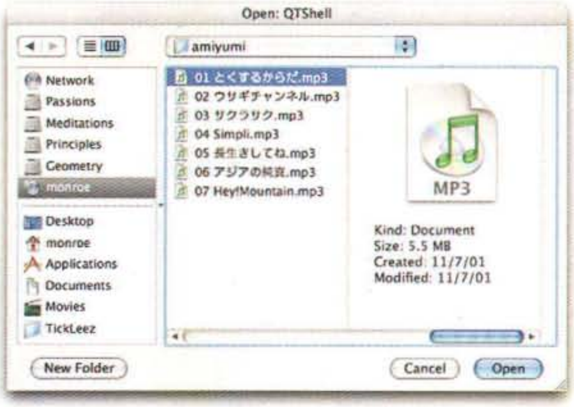


***Figure 3:** The file-opening dialog box*

In Navigation Services 3.0, this scheme was changed significantly but not enough to cause major upheavals in our existing source code. We still need to get the default options for the dialog box, but now we need to call NavGetDefaultDialogCreationOptions, not NavGetDefaultDialogOptions:

```
NavGetDefaultDialogCreationOptions(&myOptions);
myOptions.optionFlags -= kNavNoTypePopup;
myOptions.optionFlags -= kNavAllowMultipleFiles;
myOptions.modality = kWindowModalityAppModal;
myOptions.clientName = CFStringCreateWithPascalString(NULL,
  gAppName, GetApplicationTextEncoding());
```

This departs from our existing code in several ways. First, the clientName field is a CFString, not a Pascal string. We can create that string from an existing Pascal string by calling CFStringCreateWithPascalString. Later we'll need to release the string like this:

```
CFRelease(myOptions.clientName);
```

The other interesting option is the modality field, which can take these values:

```
enum {
  kWindowModalityNone              = 0,
    kWindowModalitySystemModal      = 1,
  kWindowModalityAppModal           = 2,
  kWindowModalityWindowModal      = 3
};
```

As you can see, we use the kWindowModalityAppModal constant, which causes the dialog box to prevent user interaction with all other windows in the application. A sheet would use the kWindowModalityWindowModal constant, which blocks user interaction with just one other window (the one the sheet is attached to).

Once we've set up the dialog box options, we create the dialog box by calling NavCreateGetFileDialog:

```
myErr = NavCreateGetFileDialog(&myOptions, NULL,
    myEventUPP, NULL, (NavObjectFilterUPP)theFilterProc,
    (void*)myOpenList, &myDialogRef);
```

This call however does not display the dialog box to the user. This gives us an opportunity to further customize the appearance of the dialog box by calling NavCustomControl (as we'll do in a few moments). Once we've customized the box to our liking, we show it to the user by calling NavDialogRun.

When NavDialogRun returns, we can call NavDialogGetReply to retrieve a NavReplyRecord record that contains information about the selected file. We then proceed as before by getting an FSSpec for the selected file, which we return to the caller. Listing 1 shows the new definition of QTFrame_GetOneFileWithPreview.

Listing 1: Eliciting a movie file from the user

QTFrame_GetOneFileWithPreview

```
OSErr QTFrame_GetOneFileWithPreview (short theNumTypes,
    QTFrameTypeListPtr theTypeList, FSSpecPtr theFSSpecPtr,
    void *theFilterProc)
{
#if TARGET_OS_WIN32
  StandardFileReply         myReply;
#endif
#if TARGET_OS_MAC
  NavDialogRef                myDialogRef = NULL;
  NavReplyRecord              myReply;
  NavTypeListHandle         myOpenList = NULL;
  NavEventUPP               myEventUPP =
                   NewNavEventUPP(QTFrame_HandleNavEvent);
  NavDialogCreationOptions myOptions;
#endif
  OSErr                       myErr = noErr;

  if (theFSSpecPtr == NULL)
    return(paramErr);

  // deactivate any frontmost movie window
  QTFrame_ActivateController(QTFrame_GetFrontMovieWindow(),
    false);

#if TARGET_OS_WIN32
  // prompt the user for a file
  StandardGetFilePreview((FileFilterUPP)theFilterProc,
      theNumTypes, (ConstSFTypeListPtr)theTypeList,
      &myReply);
   if (!myReply.sfGood)
      return(userCanceledErr);

   // make an FSSpec record
   myErr = FSMakeFSSpec(myReply.sfFile.vRefNum,
         myReply.sfFile.parID, myReply.sfFile.name,
         theFSSpecPtr);
#endif

#if TARGET_OS_MAC
   // specify the options for the dialog box
   NavGetDefaultDialogCreationOptions(&myOptions);
   myOptions.optionFlags -= kNavNoTypePopup;
   myOptions.optionFlags -= kNavAllowMultipleFiles;
   myOptions.modality = kWindowModalityAppModal;
   myOptions.clientName = CFStringCreateWithPascalString
               (NULL, gAppName, GetApplicationTextEncoding());

   // create a handle to an 'open' resource
   myOpenList = (NavTypeListHandle)
         QTFrame_CreateOpenHandle(kApplicationSignature,
            theNumTypes, theTypeList);
   if (myOpenList != NULL)
      HLock((Handle)myOpenList);

   // prompt the user for a file
   myErr = NavCreateGetFileDialog(&myOptions, NULL,
         myEventUPP, NULL, (NavObjectFilterUPP)theFilterProc,
         (void*)myOpenList, &myDialogRef);
   if ((myErr == noErr) && (myDialogRef != NULL)) {
      AEDesc            myLocation = {typeNull, NULL};

      // if no open-file location exists, use ~/Movies
      if (QTFrame_GetCurrentFileLocationDesc(&myLocation,
         kGetFileLoc) == noErr)
         NavCustomControl(myDialogRef, kNavCtlSetLocation,
            (void *)&myLocation);

      myErr = NavDialogRun(myDialogRef);
      if (myErr == noErr) {
         myErr = NavDialogGetReply(myDialogRef, &myReply);
         if ((myErr == noErr) && myReply.validRecord) {
            AEKeyword   myKeyword;
            DescType      myActualType;
            Size          myActualSize = 0;

            // get the FSSpec for the selected file
            if (theFSSpecPtr != NULL)
               myErr = AEGetNthPtr(&(myReply.selection), 1,
                  typeFSS, &myKeyword, &myActualType,
                  theFSSpecPtr, sizeof(FSSpec), &myActualSize);

            NavDisposeReply(&myReply);
         }
      }

         NavDialogDispose(myDialogRef);
   }

   // clean up
   if (myOpenList != NULL) {
      HUnlock((Handle)myOpenList);
      DisposeHandle((Handle)myOpenList);
   }

   if (myOptions.clientName != NULL)
      CFRelease(myOptions.clientName);

   DisposeNavEventUPP(myEventUPP);
#endif

   return(myErr);
}
```

Choosing a Filename to Save

The changes required to upgrade our existing file-selection routine QTFrame_PutFile are entirely analogous to those considered in the previous section. We need to replace NavPutFile by the combination of NavCreatePutFileDialog, NavDialogRun, NavDialogGetReply, and NavDialogDispose. There is only one "gotcha" here, and it's a big one: the FSSpec that we get when we call AEGetNthPtr no longer specifies the file we want to save the movie into (as it did with NavPutFile); rather, it specifies the *directory* that contains the file. I'm guessing that this was changed to better support values of type FSRef, which cannot specify non-existent files. The preferred way to respond to NavDialogGetReply is apparently to ask for the parent directory of the chosen filename in the form of an FSRef and then to create the file by calling FSRefCreateFileUnicode, which takes the parent directory and a Unicode filename. Since we are retaining our dependence on FSSpec values, we need to jump though a hoop or two.

What we need to do is find the directory ID of the parent directory returned to us, so that we can create an FSSpec record for the chosen file itself. **Listing 2** shows some File Manager voodoo that accomplishes this.

Listing 2: Finding the directory ID of a file's parent directory

QTFrame_PutFile

```
myErr = AEGetNthPtr(&(myReply.selection), 1, typeFSS,
            &myKeyword, &myActualType, &myDirSpec,
            sizeof(FSSpec), &myActualSize);
if (myErr == noErr) {
   myFileName = NavDialogGetSaveFileName(myDialogRef);
   if (myFileName != NULL) {
      CInfoPBRec      myPB;

      myPB.dirInfo.ioVRefNum = myDirSpec.vRefNum;
      myPB.dirInfo.ioDrDirID = myDirSpec.parID;
      myPB.dirInfo.ioNamePtr = myDirSpec.name;
      myPB.dirInfo.ioFDirIndex = 0;
      myPB.dirInfo.ioCompletion = NULL;

      myErr = PBGetCatInfoSync(&myPB);
      if (myErr == noErr) {
         CFStringGetPascalString(myFileName, myString,
               sizeof(FSSpec), GetApplicationTextEncoding());
         myErr = FSMakeFSSpec(myPB.dirInfo.ioVRefNum,
               myPB.dirInfo.ioDrDirID, myString, &myMovSpec);
         if (myErr == fnfErr)
            myErr = noErr;
      }

      if (myErr == noErr)
         *theFSSpecPtr = myMovSpec;
   }
}
```

The trick here is to know that on entry to the PBGetCatInfoSync call, the ioDrDirID field should be set to the directory ID of the parent directory of the directory containing the chosen file, which is what is contained in the FSSpec returned by AEGetNthPtr; on exit that field will contain the directory ID of the directory itself (not its parent). Once we've retrieved that directory ID, we can then call FSMakeFSSpec to create an FSSpec for the file itself.

Showing the Save Changes Dialog Box

QTShell uses one other Navigation Services function, NavAskSaveChanges, in the Macintosh version of the QTFrame_DestroyMovieWindow function. We need to replace this with the newer NavCreateAskSaveChangesDialog. Listing 3 shows the key changed portions of QTFrame_DestroyMovieWindow.

```
    myLocKey = CFSTR("AppleNavServices:PutFile:0:Path");
  else
    myLocKey = CFSTR("AppleNavServices:GetFile:0:Path");

  // see whether our application's Preferences plist already contains a file location
  myLoc = CFPreferencesCopyAppValue(myLocKey,
                              kCFPreferencesCurrentApplication);
  if (myLoc != NULL) {
    // there is an existing location
    CFRelease(myLoc);
    myErr = paramErr;
  } else {
    // there is no existing location; return a descriptor for ~/Movies
    myErr = FSFindFolder(kUserDomain,
      kMovieDocumentsFolderType, kCreateFolder, &myFSRef);

    if (myErr == noErr)
      myErr = FSGetCatalogInfo(&myFSRef, kFSCatInfoNone,
                    NULL, NULL, &myFSSpec, NULL);

    if (myErr == noErr)
      myErr = AECreateDesc(typeFSS, &myFSSpec,
                    sizeof(FSSpec), theLocation);
  }

  return(myErr);
}
#endif
```

All that remains is to call this function inside of QTFrame_GetOneFileWithPreview and QTFrame_PutFile. If you look back at Listing 3, you'll see these lines of code immediately preceding the call to NavDialogRun:

```
if (QTFrame_GetCurrentFileLocationDesc(&myLocation,
        kGetFileLoc) == noErr)
  NavCustomControl(myDialogRef, kNavCtlSetLocation,
        (void *)&myLocation);
```

Movie Storage Functions

QuickTime 6.0 introduced a set of functions called the *movie storage* APIs. The fundamental idea here is dead simple: instead of being restricted to opening, updating, creating, and deleting movie *files*, we should be able to perform these operations on any containers that hold movie data. As you know, the most general means of picking out movie data is by using a data reference. Accordingly, the movie storage APIs allow us to operate on movie data using data references and their associated data handlers.

Let's consider an example. Our application currently opens a movie specified by a file system specification record by calling OpenMovieFile, like this:

```
myErr = OpenMovieFile(&myFSSpec, &myRefNum, fsRdWrPerm);
```

If successful, OpenMovieFile returns a file reference number, which we use in all subsequent operations on the movie file. For instance, we can read the movie from that file using this code:

```
myErr = NewMovieFromFile(&myMovie, myRefNum, &myResID,
          NULL, newMovieActive, NULL);
```

When we later want to save the user's changes to a movie, we call UpdateMovieResource, passing in the file reference number and the movie resource ID:

```
myErr = UpdateMovieResource(myMovie,
     (**myWindowObject).fFileRefNum,
     (**myWindowObject).fFileResID, NULL);
```

Using the new movie storage APIs, we can use OpenMovieStorage to open movie data specified by a data reference:

```
myErr = OpenMovieStorage(myDataRef, myDataRefType,
     kDataHCanRead + kDataHCanWrite, &myDataHandler);
```

If successful, OpenMovieStorage returns an instance of a data handler, which we use in all subsequent operations on the movie container. For instance, we can save the user's changes to a movie using this code:

```
myErr = UpdateMovieInStorage(myMovie,
     (**myWindowObject).fDataHandler);
```

Here's a list of the new movie storage APIs:

CreateMovieStorage (replaces CreateMovieFile)
OpenMovieStorage (replaces OpenMovieFile)
NewMovieFromStorageOffset (replaces NewMovieFromFile)
CloseMovieStorage (replaces CloseMovieFile)
DeleteMovieStorage (replaces DeleteMovieFile)
AddMovieToStorage (replaces AddMovieResource)
PutMovieIntoStorage (replaces PutMovieIntoFile)
UpdateMovieInStorage (replaces UpdateMovieResource)
FlattenMovieDataToDataRef (replaces FlattenMovieData)

It's actually quite easy to upgrade QTShell to use these new functions. In this section, we'll see how to do this.

Maintaining Movie Storage Identifiers

First, as you probably have guessed from the snippet of code that calls UpdateMovieInStorage, we need to add a few fields to our window object record to keep track of the data reference, its type, and the data handler associated with the storage container.

```
typedef struct {
  WindowReference          fWindow;
  Movie                      fMovie;
  MovieController            fController;
  GraphicsImportComponent    fGraphicsImporter;
FSSpec                   fFileFSSpec;
  short                      fFileResID;
  short                      fFileRefNum;
  Boolean                    fCanResizeWindow;
  Boolean                    fIsDirty;
  Boolean                    fIsQTVRMovie;
  QTVRInstance               fInstance;
  OSType                     fObjectType;
  Handle                     fAppData;
#if USE_DATA_REF_FUNCTIONS
  Handle                     fDataRef;
  OSType                     fDataRefType;
  DataHandler              fDataHandler;
#endif
} WindowObjectRecord, *WindowObjectPtr, **WindowObject;
```

Notice that we use the compiler flag USE_DATA_REF_FUNCTIONS to conditionalize our code. This allows us to switch back to using the file-based functions if the need arises.

Opening a Movie

Perhaps the trickiest part of migrating to the movie storage functions is deciding how to open a movie storage container. Our file-based code calls OpenMovieFile and then NewMovieFromFile. So we might expect to call OpenMovieStorage and then NewMovieFromStorageOffset. But that's not quite right. NewMovieFromStorageOffset requires us to specify an offset to the movie atom within the storage container. In most cases we don't know what that offset is. Further, if we simply pass an offset of 0, we won't be able to open any QuickTime movie files that are not Fast Start files (where the movie atom is the first atom in the file). So we need a different strategy.

What seems to work is to call OpenMovieStorage and then NewMovieFromDataRef. Listing 5 shows a section of our revised version of QTFrame_OpenMovieInWindow.

Listing 5: Opening a movie

QTFrame_OpenMovieInWindow

```
#if USE_DATA_REF_FUNCTIONS
myErr = QTNewDataReferenceFromFSSpec(&myFSSpec, 0,
            &myDataRef, &myDataRefType);
if (myErr != noErr)
  goto bail;

// ideally, we'd like read and write permission, but we'll settle for read-only permission
myErr = OpenMovieStorage(myDataRef, myDataRefType,
            kDataHCanRead + kDataHCanWrite, &myDataHandler);
if (myErr != noErr)
  myErr = OpenMovieStorage(myDataRef, myDataRefType,
            kDataHCanRead, &myDataHandler);

// if we couldn't open the file with even just read-only permission, bail....
if (myErr != noErr)
  goto bail;

// now fetch the first movie from the file
myErr = NewMovieFromDataRef(&myMovie, newMovieActive,
            &myResID, myDataRef, myDataRefType);
if (myErr != noErr)
  goto bail;
#else
// ideally, we'd like read and write permission, but we'll settle for read-only permission
myErr = OpenMovieFile(&myFSSpec, &myRefNum, fsRdWrPerm);
if (myErr != noErr)
  myErr = OpenMovieFile(&myFSSpec, &myRefNum, fsRdPerm);

// if we couldn't open the file with even just read-only permission, bail....
if (myErr != noErr)
  goto bail;

// now fetch the first movie from the file
myResID = 0;
myErr = NewMovieFromFile(&myMovie, myRefNum, &myResID,
            NULL, newMovieActive, NULL);
if (myErr != noErr)
  goto bail;
#endif
```

Then we need to save the movie storage identifiers in our window object record, like this:

```
#if USE_DATA_REF_FUNCTIONS
  (**myWindowObject).fDataRef = myDataRef;
  (**myWindowObject).fDataRefType = myDataRefType;
  (**myWindowObject).fDataHandler = myDataHandler;
#endif
```

Saving Changes to a Movie

To save a user's changes to a movie back into its storage container, we can call UpdateMovieInStorage. **Listing 6** shows the lines we've altered in the function QTFrame_UpdateMovieFile.

Listing 6: Updating a movie's storage

QTFrame_UpdateMovieFile

```
#if USE_DATA_REF_FUNCTIONS
if ((**myWindowObject).fDataHandler == NULL)
  myErr = QTFrame_SaveAsMovieFile(theWindow);
else
  myErr = UpdateMovieInStorage(myMovie,
            (**myWindowObject).fDataHandler);
#else
if ((**myWindowObject).fFileRefNum == kInvalidFileRefNum)
  myErr = QTFrame_SaveAsMovieFile(theWindow);
else
  myErr = UpdateMovieResource(myMovie,
            (**myWindowObject).fFileRefNum,
            (**myWindowObject).fFileResID, NULL);
#endif
```

Closing a Movie

When we're finished working with a movie, we can close it by calling CloseMovieStorage. We also need to dispose of the data reference and the data handler instance associated with the movie. Listing 7 shows the changed lines in the function QTFrame_CloseWindowObject.

Listing 7: Closing a movie

QTFrame_CloseWindowObject

```
#if USE_DATA_REF_FUNCTIONS
if ((**theWindowObject).fDataHandler != NULL) {
  CloseMovieStorage((**theWindowObject).fDataHandler);
  CloseComponent((**theWindowObject).fDataHandler);
  (**theWindowObject).fDataHandler = NULL;
}

if ((**theWindowObject).fDataRef != NULL) {
  DisposeHandle((**theWindowObject).fDataRef);
  (**theWindowObject).fDataRef = NULL;
}
#else
// close the movie file
if ((**theWindowObject).fFileRefNum != kInvalidFileRefNum) {
  CloseMovieFile((**theWindowObject).fFileRefNum);
  (**theWindowObject).fFileRefNum = kInvalidFileRefNum;
}
#endif
```

CONCLUSION

Part of the price of delivering a modern QuickTime application is the inevitable need to continually upgrade its underpinnings as the operating system and user interface APIs evolve, or indeed as QuickTime itself evolves. In this article, we've seen how to use the currently recommended functions for selecting files and for opening and operating on movie data. The next step, which we have not taken here, would be to systematically replace all uses of the FSSpec data type by uses of the FSRef data type. This would give us a thoroughly modern application capable of opening movie files with Unicode or very long filenames.

By Tom Djajadiningrat & Marcelle Stienstra
Mads Clausen Institute for Product Innovation
University of Southern Denmark

Programming the BasicStamp

Using MacBS2

ABSTRACT

This article explains how to program a BasicStamp microcontroller using a Macintosh. It takes an absolute beginner's perspective and talks you through the whole process including how to hook up a programmer board, how to write some programs in the BasisStamp language PBASIC, and how to do some simple input and output such as reading switches and controlling LEDs.

INTRODUCING THE BASICSTAMP

The BasicStamp by Parallax is probably the most popular microcontroller amongst electronics hobbyists. One of the reasons for this is that it can be programmed in a simplified, customized form of BASIC called PBasic, unlike other microcontrollers such as Microchip PICs or Atmel AVRs which usually need to be programmed in C(++) or assembler. A second reason is that the development tools are free. Once you have bought a programmer board and a BasicStamp microcontroller you do not need to make any further investments in compilers or integrated development environments. Finally, the documentation and website that Parallax provide are comprehensive and written in plain language with many examples ranging from beginner's level to some pretty complex stuff. In any case, Parallax seem to consciously avoid much of the electronic engineering jargon that you find in the documentation of other microcontroller manufacturers.

You may wonder: "If this BasicStamp thing is so marvelous why isn't every electronic product powered by one?". Of course, there are drawbacks to BasicStamps. The first is speed, as a BasicStamp is basically a PIC microcontroller with a BASIC interpreter stuck on top which causes a performance penalty. The second is control. The PBASIC that the BasicStamp uses hides much complexity but also moves you one level further away from the hardware, reducing the amount of control you have over it. Last but not least, the BasicStamp is expensive, ranging from about 50 to 90 US Dollars. Compared to PICs where 10 US Dollars buys you a top model, that is expensive.

With these characteristics it is hardly surprising that the BasicStamp is mainly used in situations in which ease and speed of development are more important than cutting edge performance or high volume production with low unit cost. This is usually the case with hobbyist work but also with some one-off prototypes and sometimes even with small production runs.

BASICSTAMP ON MAC

So what has all of this got to do with the Macintosh? For a long time, there were no Macintosh development tools for the BasicStamp, requiring you to use VirtualPC if you wanted to use a Mac [1]. Recently, however, that situation changed when Parallax introduced a PBasic Tokenizer library for OSX and Murat M. Konar introduced MacBS2, an OSX native programming environment for the BasicStamp [5]. This is what you need to get stamping on the Mac:

Hardware

- Macintosh running OSX.2 or later
- A Keyspan USB to serial adapter [2]. (costs: 40 USD for a USA-19, 60 USD for a USA-28x)
  You can either use a USA-28x (**Figure 1**) with two traditional Macintosh style Mini-Din8 connectors or a USA-19 (**Figure 2**) with a single PC style DB9 connector. The advantages of the USA-19 are that it is the least expensive Keyspan USB-serial adapter and that you don't have to modify and solder a serial cable as you can simply use a standard DB9 male-DB9 female cable. The advantages of the USA-28x are that you get two serial ports and that you may regain connectivity with legacy Macintosh serial peripherals. In a minute we will give you an explanation on how to make a suitable serial cable should you wish to use a USA-28x.

Tom Djajadiningrat and **Marcelle Stienstra** work at the Mads Clausen Institute for Product Innovation, a research and teaching facility of the University of Southern Denmark. When not gossiping about their colleagues on ICQ, they claim to teach interaction design to students on the IT Product Design masters course. You can reach them on j.p.djajadiningrat@mci.sdu.dk and marcelle@mci.sdu.dk.

***Figure 1:** Keyspan USA-28x USB-serial adapter with two Mac-style Mini-Din8 connectors*

***Figure 2:** Keyspan USA-19 USB-serial adapter with a single PC-style DB9 connector*

- A BasicStamp 2 Variant (costs: 50-90 US Dollars)

There are a number of BS2 microcontrollers which differ in speed, number of I/O pins, supported calls and obviously price [3]. We chose a BS2p24 which is the fastest Stamp currently available (**Figure 3**). You cannot use the simpler, older and cheaper BS1, or the JAVA-based Javelin Stamp, simply because Parallax does not provide Mac-compatible tokenizer libraries for them (a tokenizer library is software that converts PBASIC source code to the "program tokens" that can be downloaded to BASIC Stamp modules). Therefore, it is not possible for MacBS2 to support these microcontrollers.

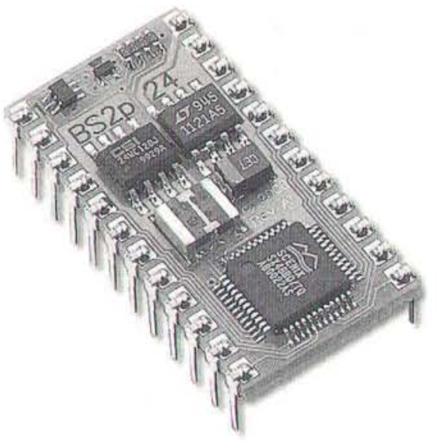


***Figure 3:** A BasicStamp. This particular one is a BS2p24*

- A BasicStamp programming board (25-200 US Dollars) Parallax makes a whole range of programming boards which differ in the number and type of connectors that are provided [4]. Some are very minimalistic, offering only a serial connector and a socket for the BasicStamp itself, others add such things as LCD connectors and breadboards (**Figure 4**). We chose one called the BS2p24 Demo Board because it has a small breadboard integrated on it which facilitates prototyping without soldering and connectors for some more advanced features of the BS2p24 such as I2C and iButton (if you don't know what these terms mean, don't worry about it).

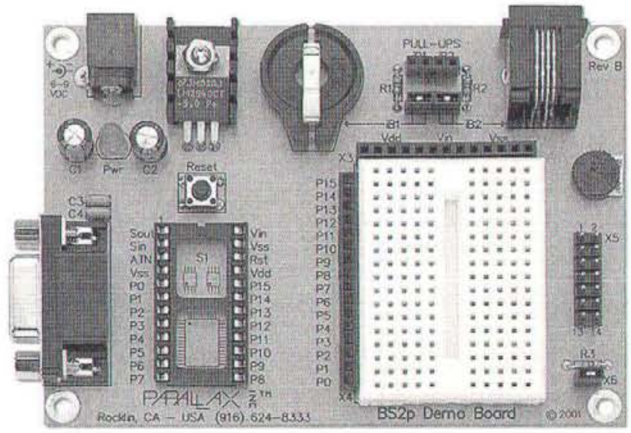


***Figure 4:** A BasicStamp programming board. In this case, a BS2p24 Demo Board*

- A Power Adapter or a 9V battery (costs: 10 US Dollars) Depending on which programmer board you use you will need either a power adapter (6-9V) or a 9V battery.

Software

- Keyspan Drivers
  Download from http://www.keyspan.com/downloads/macosx/
  Make sure you have drivers that match your version of OSX. We use 1.8 which matches OSX.3 'Panther'.
- MacBS2, Mac Tokenizer and BasicStamp Manual
  Download from http://www.muratnkonar.com/MacBS2/
  What is very elegant about MacBS2, is that it automatically downloads and installs the Parallax tokenizer and BasicStamp Manual when you need them.

Total costs

If you do the maths, you will find that the stuff required for the BasicStamp will set you back between 115 and 360 US Dollars. The setup we are using cost us approximately 250 US Dollars.

Creating a serial cable to connect a BasicStamp programmer board to a Keyspan USA-28x

(If you use a Keyspan USA-19 High Speed USB-serial adapter with a DB9 connector, you can safely skip this section: just buy a 'straight thru' male to female DB9 cable)

The information provided here is a more elaborate version for the electronically impaired of what you find in MacBS2 under the menu **Help**, item **Programming Wiring Diagram**.

Required components:

- Mini-Din8 to Mini-Din8 Macintosh serial cable
- DB9 male connector

The DB9 connector you can probably find at your local electronics store. The Mini-Din8 cable is not so current anymore so you may need to hunt around a bit. If all else fails, you can always buy one from www.maccables.com.

Instructions

To use a Keyspan USA-28x you need to make a cable with a male DB9 connector on one side and a male Mini-Din8 connector on the other. The easiest way to make one of these is to start with a Macintosh style Mini-Din8 to Mini-Din8 cable and cut it through the middle so that you end up with two cables, each of which have one end with a Mini-Din8 connector and one bare end with exposed wires. This way you don't have to solder the Mini-Din8 plug which has the Apple elegance and is therefore hell to solder and just need to make the end with the PC-style DB9 plug, which is lumpy, coarse and therefore easy to solder. The easiest way to get the cable right is:

- First, figure out which of the wires on the bare end of the cable are linked to the relevant pins of the Mini-Din8 plug. **Figure 5** shows a head-on photo of the Mini-Din8 plug. **Figure 6** shows the same view diagrammatically with the relevant pins indicated. Using a multimeter set to measure resistance, hold one probe on the pins of the Mini-Din8 plug and one on the bare wires. You are looking for the wires connected to the pins *gnd*, *sin*, *sout* and *attention*. Note down the colours of the wires. Table I shows which colours matched which pins in our case, though your cable may use different colours so tread carefully.

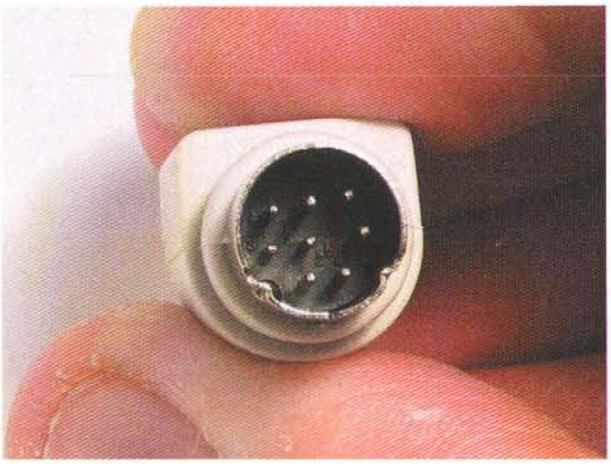


Figure 5: *The Mini-Din8 plug head-on*

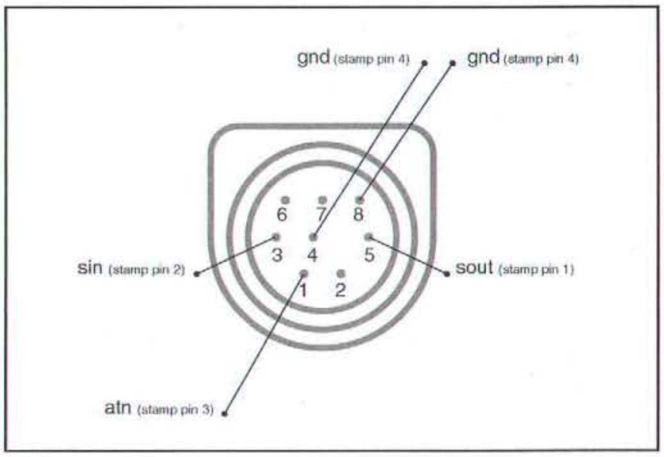


Figure 6: *The Mini-Din8 plug head-on with the relevant pins indicated*

mini-Din8	serial cable	DB9	BasicStamp	
pin#	strand colour	pin#	pin name	pin#
5	orange	2	sout	1
3	green	3	sin	2
1	red	4	atn	3
4	yellow	5	gnd	4
8	blue	5	gnd	4

Table 1: *a summary of the connections*

- Secondly, we figure out which pins of the DB9 male plug match the relevant pins. Plug the loose DB9 male connector into the DB9 female connector on your BasicStamp board so that we can measure directly on the soldering side of the male DB 9 plug (**Figure** 7 and **8**). With a multimeter, work out which pins of the DB9 plug are connected to pin 1 (sout), pin 2 (sin), and pin 4 (gnd) of the BasicStamp. Pin 3 (atn) is a bit of strange case as it is not possible to measure continuity between pin 3 on the BasicStamp and any pin on the DB9 plug. Still, even though continuity cannot be measured, pin 3 of the Stamp is connected to a pin on the DB9 plug in between gnd and sin. Looking at the soldering side of the DB9 plug, the connections are shown diagrammatically in **Figure 9**. You can refer back to **Table 1** if you get stuck.

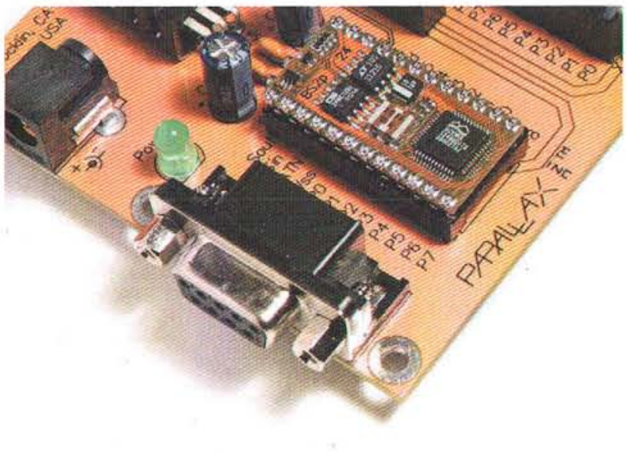


***Figure** 7: Your BasicStamp programming board has a female DB9 connector...*

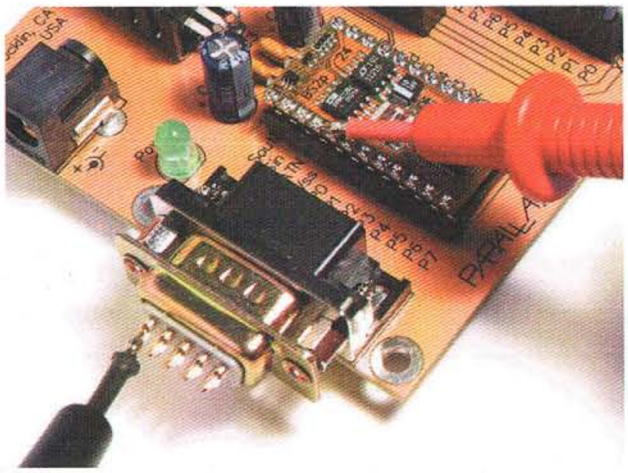


***Figure** 8: ...in which you can plug a DB9 male connector so that you can easily measure directly on the soldering side.*

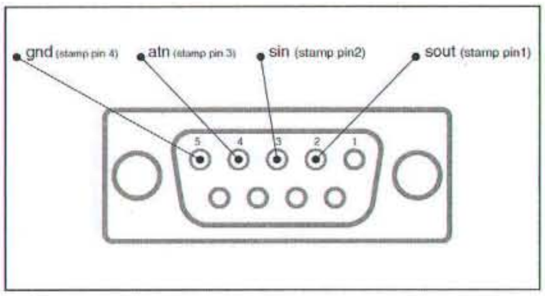


***Figure** 9: A diagram of the rear, soldering side of the DB9 plug.*

- All that is left now, is to solder the DB9 plug. Before you put on the housing of the DB9 plug do a final check all the way from those MiniDin-8 pins are really connected to the right DB9 pins. The finished soldering job is shown in **Figure 10**.

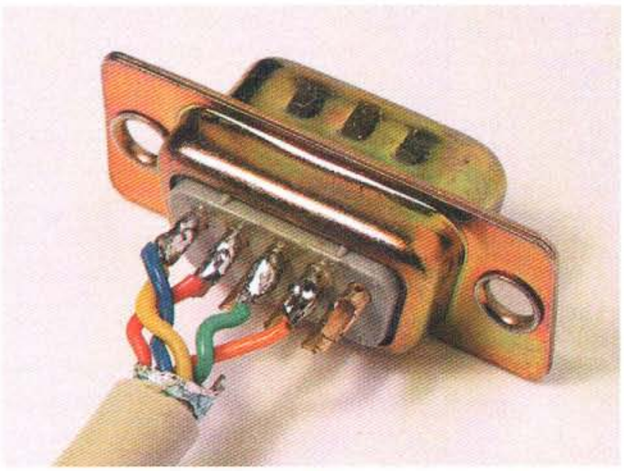


***Figure** 10: The finished soldering job on the DB9 plug*

CHECKING THE CONNECTION

- Hook up your Keyspan adapter to the USB port of your Mac.
- Do a sanity check: does the Mac acknowledge the Keyspan adapter? You can check by running the Keyspan Serial

GraphicConverter converts pictures to different formats. Also it contains many useful features for picture manipulation.

See ***www.lemkesoft.com*** for more information.

Assistant which should have ended up on your harddrive during installation of the Keyspan drivers, probably in your Applications folder. What you should see is either the dialog box in **Figure 11** or **Figure 12** depending on which model Keyspan adapter you use.

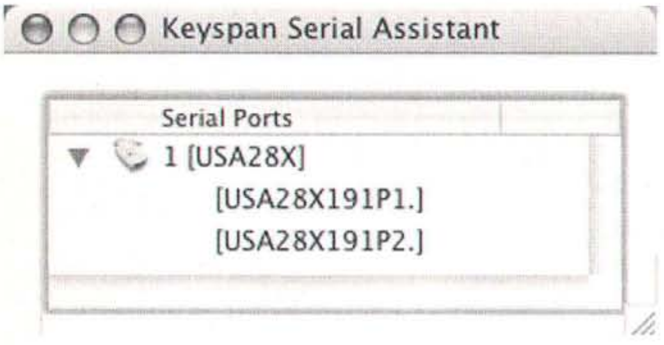


Figure 11: *The Keyspan Serial Assistant acknowledging a USB-28x with two serial ports*

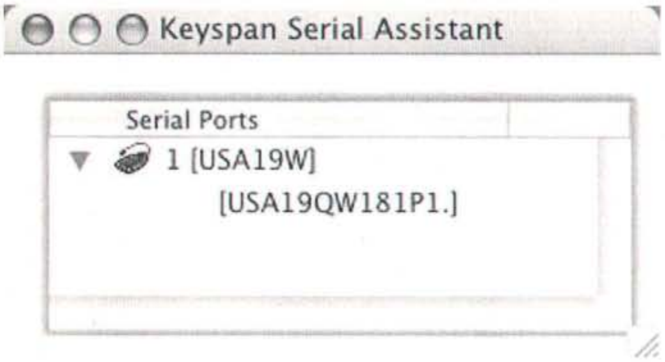


Figure 12: *The Keyspan Serial Assistant acknowledging a USB-19 with one serial port*

- Plug your BasicStamp into your programmer board.
- Connect the power supply or the 9V battery to the programmer board. The power LED on the board should light up.
- Now connect your Keyspan adapter to the programmer board with your serial cable.
- Start up MacBS2. In the pop-up menu in the windows menubar you should see either one serial port for a Keyspan USA-28x (**Figure 13**) or two for a Keyspan USA-19 (**Figure 14**).

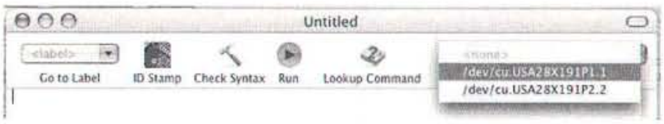


Figure 13: *The port popup menu in MacBS2 for a Keyspan USA-28x*

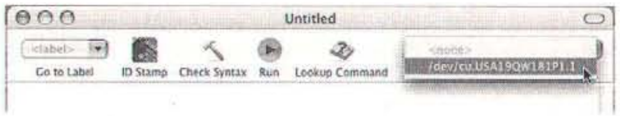


Figure 14: *The port popup menu in MacBS2 for a Keyspan USA-19*

- Click the **ID Stamp** button in the toolbar. The status bar at the bottom of the MacBS2 window says something like: "Found Basic Stamp 2p, firmware version = 1.2". Of course, the details may vary according to the type of Stamp and firmware you are using.

No joy? Check your physical connections, check that the programmer board is powered, make sure that you have correctly inserted the BS2 microcontroller and verify that you have the latest Keyspan drivers.

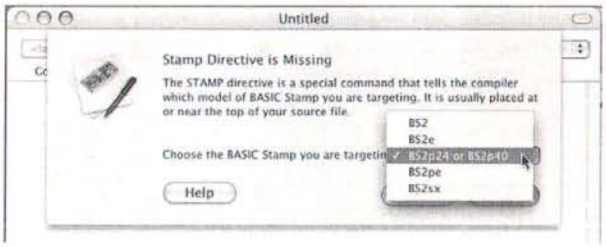
Eight Steps to Becoming a PBasic Guru

Time to do some Stamping! The easiest way to become familiar with the code-upload-debug cycle is to simply work your way through some small PBasic programs. Here is a set of eight, which will familiarize you with printing to the console window, variables, conditionals, loops and subroutines. Whether you save each program and open a new MacBS2 window for the next one and or simply modify the previous one is up to you.

1. Debug

At the start of every PBasic program, you need to tell MacBS2 which version of the BS2 you are programming. The easiest way to do this is to click the **Check Syntax** button. This brings up a dialog box with a popup menu, allowing you to choose between different models BS2 (**Figure 15**). Pressing the **Fix** button causes MacBS2 to insert a Stamp directive at the top of your file. We use a BS2p24 Stamp and therefore choose a BS2p from the popup menu causing MacBS2 to insert the code:

```
'{$STAMP BS2p}
```


Stamp Directive is Missing

The STAMP directive is a special command that tells the compiler which model of BASIC Stamp you are targeting. It is usually placed at or near the top of your source file.

Choose the BASIC Stamp you are targetin

Figure 15: *Choosing the type of BasicStamp*

In all other cases, the ' (apostrophe) starts a comment, causing MacBS2 to ignore the rest of the line. Strangely enough, the stamp directive is essential and still starts with an apostrophe. Basically, it is a kind of include statement, telling MacBS2 which kind of stamp we are addressing. Complete the program as per **Listing 1.**

Listing 1: Debug

Debug

```
'{$STAMP BS2p}

'Our first PBasic program
DEBUG "Hello World!"
```

Now, press the check syntax button (or command-K) and then the run button (or command-R). The text "Hello World!" should appear once in the debugger pane. You may have noticed that some of the code is in lower and some is in uppercase. This does not actually matter: PBasic code is not case sensitive. It does make it a little easier to decipher the code when you write PBasic reserved words in capitals and everything else in lowercase (Unlike the PC version, MacBS2 does not automatically convert PBasic reserved words to uppercase).

If you have a reset switch on your BasicStamp programming board, pressing it causes the program to run from the beginning. So in this case, everytime you press the reset button, it writes "Hello World!" to the debugger pane.

This may not be the most convincing of BasicStamp programs: how can we be sure that the Mac is not simply printing the string directly to the debugger pane? Well, unplug the serial cable from the BasicStamp programming board, press the reset button on your board to run your program from the beginning, and you'll see that the printing to the debugger pane does not happen. Plugging the serial cable back in restores the programs functionality. So, the program does actually run on the Stamp and sends the string over the serial connection to the Mac.

2. Variable

Open a new MacBS2 window and type in the code in **Listing 2.**

Listing 2: Variable

Variable

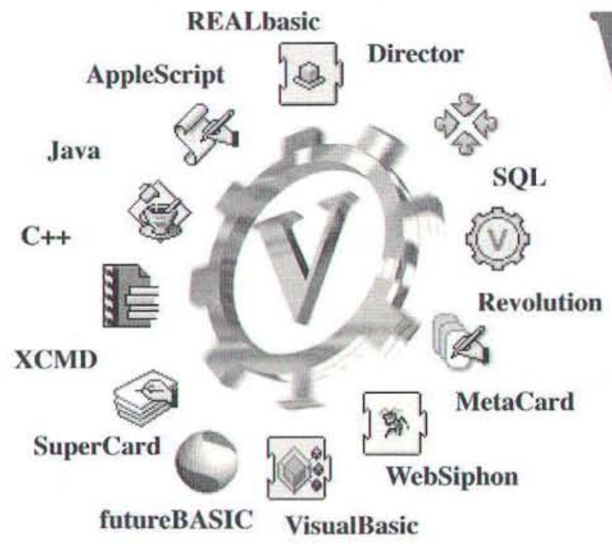
```
'{$STAMP BS2p}

' variable declaration
gNumber   VAR  BYTE

' variable initialization
gNumber = 42

'print gNumber to the debugger pane
DEBUG DEC gNumber, CR
DEBUG DEC ? gNumber
```

Press the **check syntax** button, then the **run** button. We start again with the stamp directive. What follows is a variable declaration: first the name of the variable (gNumber), then the reserved word VAR and finally the amount of memory that we want the Stamp to set aside for the variable, in this case a byte. This means that the variable can range from 0-255 in decimal, which can therefore easily store the number 42. All variables in PBasic are global. Just to remind you, we start the variable name with the letter g. We then initialize the variable gNumber with the value 42. Finally, we print the variable gNumber to the debugger pane in two ways. The first simply prints the value of the variable in decimal resulting in 42 and the carriage return forces the entry point to the next line. The second uses the DEC ? notation which makes the Stamp precede the value of the variable in decimal by variableName = and automatically follows it with a carriage return, resulting in:

```
gNumber = 42
```

3. Goto

Ah, the evil goto. In PBasic you need to create a label ending with a colon before you can use it as an argument in a goto statement. **Listing 3** shows a simple code example.



Valentina

Object-Relational SQL Database

The fastest database engine for MacOS/Windows

It operates 100's and sometimes a 1000 times faster than other systems

www.paradigmasoft.com

Hosted by macserve.net

Download full featured evaluation version

Get more out of your Macintosh!

DevDepot sells the tools, toys and technology to put more muscle into your Mac. Innovative upgrades, essential accessories, and cutting edge new products for those of you who just want more.

Visit the Official Store, managed by DevDepot, for show specials and great new products from all over the show floor!

DevDepot has it all!

More Storage

500 GB on your Desktop

The BigDrive from LaCie, FireWire 800 and all the capacity you can ask for.

Put 80 GB in your PowerBook

Why leave it behind? Keep all the data and applications youneed on your PowerBook or iBook.

Add a SuperDrive

An internal SuperDrive does it all – DVD-R/RW and CD-R/RW in one drive. iDVD compatible, easily to install.

Lucid FireWire Powered CD-R/DVD-R

Lucid CD-R/RW and DVD-R/RW drives are iTunes compatible and bus powered. Portable and Powerfull!

Better Sound

PowerWave USB Amp

Audio interface and amplifier for your Mac or PC, connect your home speakers or sound system to your USB Mac.

Noise Reduction Headphones

With airline adapter, the perfect companion for iPod and traveling with your PowerBook.

More Speed

HARMONi for iMac
Pump your original iMac to 600 MHz G4 and add FireWire!

G4 Upgrades
Put up to a 1.2 GHz G4 in your box. Upgrades for your G4 Tower, your iMac, Blue & White, Beige G3, 8500 and before!

Macintosh RAM
RAM upgrades at great prices, designed and built for best performance and 100% compatibility with your Macintosh.

Wallstreet G4
Your Wallstreet G3 PowerBook can have a 500 MHz G4 Processor!

New Interfaces

PowerMate Analog Control
Control iTunes or DVD Player, edit home movies or just use it next to your mouse as the 'ultimate' scroll wheel!

Wireless Mice
Finally a wireless mouse for Mac, done right! Portable version for PowerBook users or desktop version!

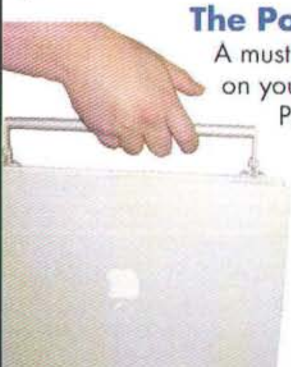
iceCAD
Ideal tool for Apple InkWell for handwriting recognition, freehand sketches, or use with graphic applications.

AirPort USB Adapter
Add AirPort connectivity to your iMac, PowerMac or any USB Macintosh!

PowerBook & iPod Innovations

Replacement Batteries
Long flight or multiple meetings, do you need more juice? Batteries for G3 (Pismo, Lombard, Wallstreet) and G4 PowerBooks!

iTrip
The ultimate iPod accessory, plugs into your iPods FireWire and Audio ports. FM broadcaster, powered and controled throug' your iPod.



The PowerBook Handle
A must have for G4 PowerBook users, put a handle on your PowerBook and great to prop up your PowerBook for a comfortable tilt.

Better AC Adapters
The smallest, most compact AC adapter for PowerBooks and iBooks. One adapter for home, one for the office!

DEV DEPOT®

www.devdepot.com

877-DEPOT-NOW

Outside US/Canada: 805-494-9797 • Fax: 805-494-9798 • orders@devdepot.com • PO Box 5200 • Westlake Village, CA 91359-5200

Listing 3: Goto

Goto

```
'{$STAMP BS2p}

loop:

  DEBUG "Hello World!", CR

  GOTO loop
```

Press the syntax button, then the run button. This program simply causes the Stamp to write "Hello World!" to the debug pane endlessly.

4. For/next

A for/next statement in PBasic looks like **Listing 4.**

Listing 4: For/next

For/next

```
'{$STAMP BS2p}

' declare a variable and reserve a nibble of memory
gCounter VAR  NIB

for gCounter = 1 to 3
  debug DEC ? gCounter
next
```

Press the **check syntax** button, then the **run** button. This prints the variable gCounter to the debug pane three times. Note that the amount of memory reserved for gCounter is a nibble (NIB) or half a byte. Yes, that is how thrifty you need to be with memory on a microcontroller: you need gCounter to count up to only three. A nibble ranges from 0-15 which is more than sufficient, so why use a full byte?

5. Out and pause

Finally, we are ready to connect some electronic components and make something happen in 'the real world'. What you need is an LED and a 330Ohm resistor. The schematic of the circuit is shown in **Figure 16. Figure 17** shows how to connect this on the breadboard of a BS2p24 Demo Board. The LED is connected to pin 3.

Type in the code in **Listing 5.**

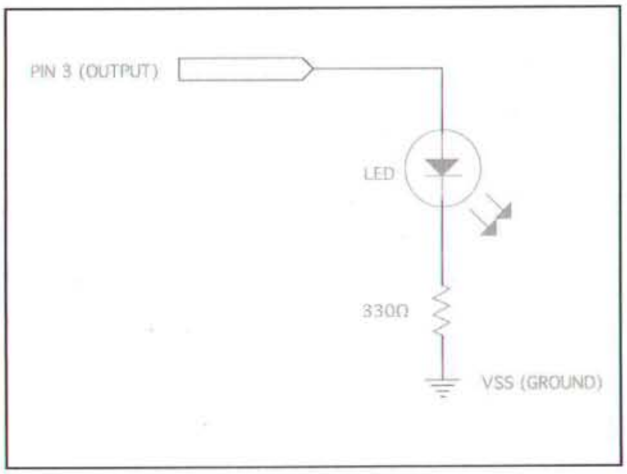


***Figure 16:** An LED connected on pin 3*

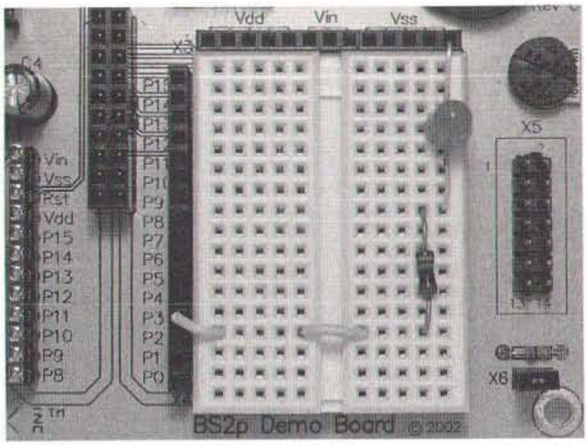


***Figure 17:** What things look like on the breadboard*

Listing 5: Out and pause

Out and pause

```
'{$STAMP BS2p}

OUTPUT 3

loop:

  OUT3 = 1
  DEBUG "LED on", CR
  PAUSE 500

  OUT3 = 0
  DEBUG "LED off", CR
  PAUSE 1000
  GOTO loop
```

The new commands in here are OUTPUT, OUT and PAUSE. OUTPUT 3 makes pin 3 act as a digital output which can either be 0 or 1. OUT3 actually sets pin 3 to 0 or 1. PAUSE takes an argument between 0 and 65535 causing the BasicStamp to wait for that many microseconds. So the PAUSE 500 causes the Stamp to delay for 0.5s whilst PAUSE 1000 causes a delay of a full second.

Check the syntax and run the program. The LED should blink in a pattern of on for 0.5 second and off for 1 second.

No joy? Remember that LEDs are directional: current only flows from the anode (+) to the cathode (-). The cathode is usually indicated by a flat side on the LED's housing or by a shorter lead. In this case we have chosen to connect the cathode to ground and the anode to the resistor.

Confused about resistor values? There is a very handy web-based resistor calculator available [6].

6. Button

You are now ready to try your hands at some interactive stuff: press a button and an LED will light up. Let's try using a pin as an input. Add the circuit as in the schematic in **Figure 18** to what you already built in **Figure 17.** Our breadboard ended up looking like **Figure 19.** Type in the code in **Listing 6.**

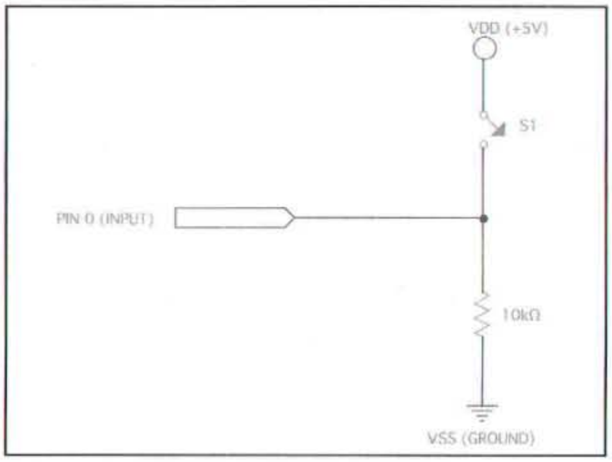


Figure 18: *The active-high button circuitry*

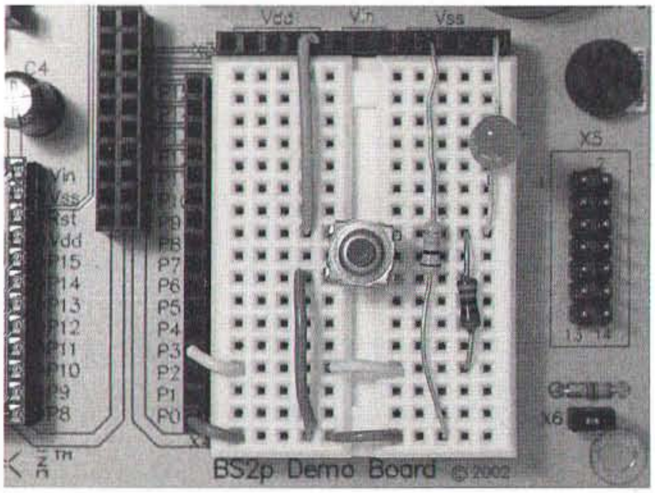


Figure 19: *The LED output and button input circuitry together*

Listing 6: Button

Button

```
'{$STAMP BS2p}

gWorkSpace    VAR  BYTE

'make pin 3 an output
OUTPUT 3


loop:

  'BUTTON Pin, DownState, Delay, Rate, Workspace,
  'TargetState,Address
  BUTTON 0, 1, 0, 0, gWorkSpace, 1, press
  DEBUG "no, not pressed", CR
  OUT3=0
  goto loop


press:

  DEBUG "yes, pressed", CR
  OUT3=1
  GOTO loop
```

The only new PBasic call in this listing is BUTTON. BUTTON is a conditional depending on the binary state of an input. It thus forms a kind of convenient mix of monitoring an input and an IF/THEN statement. It always needs to be in a loop and comes with no less than seven parameters.

The first, Pin, indicates which pin of the Stamp we are looking at. The second, Downstate tells the Stamp for which logic state we are looking for at the pin. Delay and Rate are best explained together. The Button command makes it possible to autorepeat, similar to what happens when you keep a key of your Mac pressed. The Delay parameter indicates how long it takes before autorepeat starts whilst the Rate parameter determines the intervals between autorepeats. Both parameters are expressed in the number of cycles of the loop containing BUTTON. Perhaps a more important use of Delay is the debounce feature, which prevents the BUTTON statement being triggered multiple times due to scraping contacts of mechanical switches This is useful when you are looking for a single button press. To activate debounce, you use Delay = 255. To deactivate debounce, you use Delay = 0. In this example, we use neither debounce nor autorepeat.

Workspace is a byte variable for use by BUTTON. Basically, if you declare it once and never use it for anything but the BUTTON command, everything will be ok. The sixth parameter, TargetState, determines on which state of the button we want the branch to occur. 0 = not pressed and 1 = pressed. Beware of the

high quality - competitive rates - 16 years experience - award winning

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

Full Spectrum Software

Development & Testing

Device Drivers

Carbon **Cocoa**

Real Basic

Rescue Missions

Cross Platform Development

1661 Worcester Road
Framingham, MA 01701

508-620-6400
www.FullSpectrumSoftware.com

reliable - high quality - competitive rates - efficient - award winning - qa services - reputable

competitive rates - 16 years experience - award winning - high quality difference between the DownState and TargetState parameters. Things can get confusing as the behaviour also depends on whether the switch circuitry is active high or active low and on whether you use a push-to-make or push-to-break switch.

Here we have chosen what we think is the easiest situation to get your head round: we use a push-to-make switch with active high circuitry, which therefore causes a logic high on the input pin when pressed. This means that we use DownState = 1. Also, we find it helps readability of the code to branch when the button is pressed and to continue when the button is not pressed and for that we need TargetState = 1.

Ofcourse, other combinations of circuitry, switches, DownState and TargetState can work fine too, just make sure you think things through carefully to avoid strange behaviour.

7. If/Then

Type in the code in **Listing 7.**

Listing 7: If/Then

If/Then

```
'{$STAMP BS2p}

gWorkSpace      VAR BYTE
gButtonCounter  VAR NIB

gButtonCounter = 0

loop:

  BUTTON 0,1,255,0,gWorkSpace,1,Press

noPress:

  DEBUG "no, not pressed. ", DEC ? gButtonCounter

  GOTO loop


press:

gButtonCounter = gButtonCounter + 1
DEBUG "yes, pressed. ", DEC ? gButtonCounter

if gButtonCounter = 3 then lastMessage
GOTO loop

lastMessage:
DEBUG "Button pressed 3 times"
```

The program counts button presses and ends when the user has pressed the button three times. Note that this time we use the BUTTON command with Delay = 255, meaning that debouncing is on and autorepeat off. This is ideal for detecting a single button press.

8. Gosub

One more step and you will have reached your PBasic Guruship. This example shows how to use subroutines in PBasic. Type in the code in **Listing 8.**

Listing 8: Gosub

Gosub

```
'{$STAMP BS2p}

gCounter VAR NIB

FOR gCounter = 1 to 3

  ' calling a subroutine
  GOSUB debugCounterValue

NEXT

STOP


' example of a subroutine
debugCounterValue:

  DEBUG "start subroutine", CR
  DEBUG DEC ? gCounter
  DEBUG "end subroutine", CR, CR

  RETURN
```

A subroutine needs to start with a label and end with a RETURN statement. GOSUB causes a jump to a label of that name. The code in the subroutine is then executed and the RETURN statement causes the Stamp to jump back to the line following the GOSUB.

Note the use of the STOP command after the main program. It prevents us running into the subroutine accidentally after the main loop has executed three times.

Conclusion

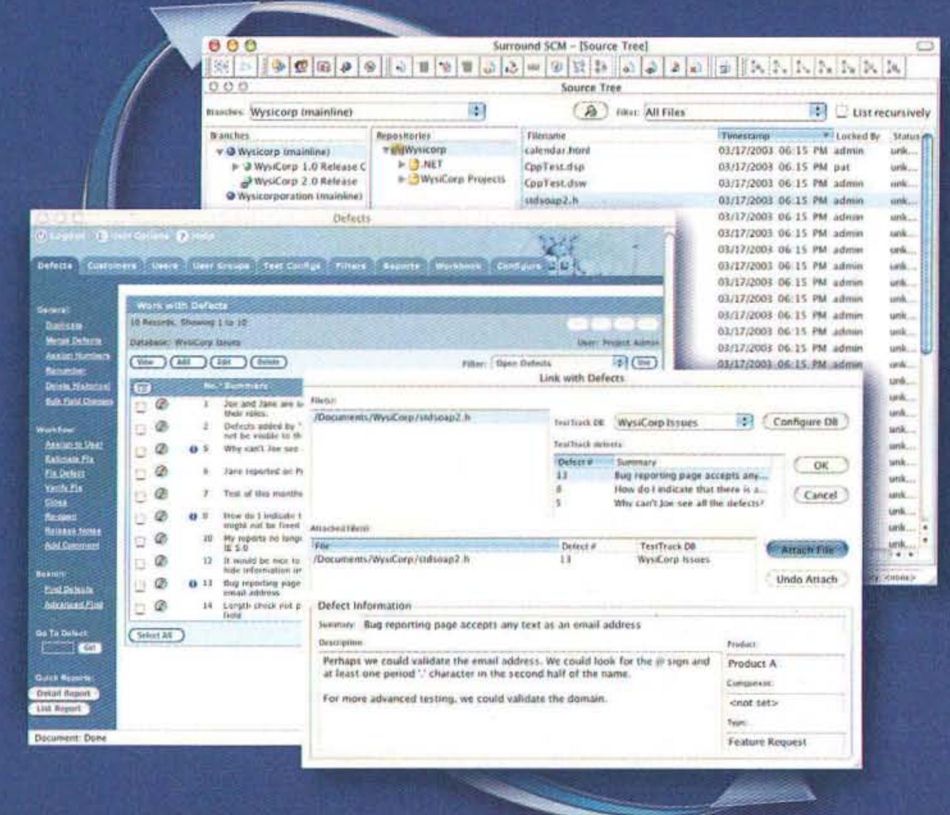
So there you have it. You can now program a BasicStamp on your Mac and do some simple input and output. Try to address some other pins using multiple LEDs or buttons. The BasicStamp Manual is very friendly, so once you start playing with a Stamp you'll quickly find yourself exploring new commands and syntax. The Parallax website has links to some great application notes from Nuts & Volts magazine in which you can find info on hooking up various sensors, servos, stepper motors etc [7]. Enjoy!

References

1. If for some incomprehensible reason you insist on using Windows-based BasicStamp development tools under VirtualPC emulation, here is how: http://www.robotstore.com/download/Stamps_on_iMac_2.13.pdf
2. Keyspan make USB-serial adapters: http://www.keyspan.com
3. Here you find a full list of all BasicStamp models: http://www.parallax.com/html_pages/products/basicstamps/basic_stamps.asp
4. Here you find a list of all the programmer boards available: http://www.parallax.com/html_pages/products/boards/programming_boards.asp
5. This is the home of the MacBS2 software: http://www.muratnkonar.com/MacBS2/
6. There is a handy JavaScript-based calculator floating around the net for working out resistor values from colour codes. Here is one site that has it: http://www.dannyg.com/examples/res2/resistor.htm
7. Here you find the Nuts & Volts application notes, ranging from very basic to some pretty hairy stuff: http://www.parallax.com/html_pages/downloads/nvcolumns/Nuts_Volts_Downloads.asp

Complete Source Control and Defect Management for Mac OS X

Seapine Software™
changing the world of software development


Surround SCM – [Source Tree]

Source Tree

Branches: Wysicorp (mainline) Filter: All Files ☐ List recursively

Branches	Repositories
Wysicorp (mainline)	Wysicorp
WysiCorp 1.0 Release C	.NET
WysiCorp 2.0 Release	WysiCorp Projects
Wysicorporation (mainline)	

Filename	Timestamp	Locked By	Status
calendar.html	03/17/2003 06:15 PM	admin	unk...
CppTest.dsp	03/17/2003 06:15 PM	pat	unk...
CppTest.dsw	03/17/2003 06:15 PM	admin	unk...
stdsoap2.h	03/17/2003 06:15 PM	admin	unk...

Defects

Work with Defects

Filter: Open Defects

Link with Defects

File(s): /Documents/WysiCorp/stdsoap2.h

TestTrack DB: WysiCorp Issues — Configure DB

TestTrack defects

Defect #	Summary
13	Bug reporting page accepts any...
8	How do I indicate that there is a...
5	Why can't Joe see all the defects?

Attached file(s)

File	Defect #	TestTrack DB
/Documents/WysiCorp/stdsoap2.h	13	WysiCorp Issues

Defect Information

Summary: Bug reporting page accepts any text as an email address

Description: Perhaps we could validate the email address. We could look for the @ sign and at least one period '.' character in the second half of the name.

For more advanced testing, we could validate the domain.

Product: Product A
Component: <not set>
Type: Feature Request

Effective source code control and defect tracking require powerful, flexible, and easy-to-use tools—Surround SCM and TestTrack Pro

- Complete source code control with private workspaces, automatic merging, role-based security, and more
- Comprehensive defect management — track bug reports and change requests, define workflow, customize fields
- Fast and secure remote access to your source files and defects — work from anywhere
- Advanced branching simplifies managing multiple versions of your products
- Link code changes with defects and change requests — know who changed what, when, and why
- Scalable and reliable cross-platform, client/server solutions support Mac OS X, Windows, Linux, and Solaris
- Exchange data using XML and ODBC, extend and automate with SOAP support
- Licenses priced to fit your budget

Seapine Software Product Lifecycle Management
Award winning, easy-to-use software development tools

Seapine **Surround SCM**

Seapine **TestTrack** *PRO*

software development 13th annual product excellence award JOLT

Download Surround SCM and TestTrack Pro at www.seapine.com or call 1-888-683-6456

by Benjamin S. Waldie

User Interaction Basics

Last month, we looked at some of the features of the new *Script Editor*, which was released with *Mac OS X Panther (10.3)*. Now we are going to get started with actually writing some AppleScript code! This month's article will explain how, with only minimal code, you can update your AppleScripts to interact with the user. We will primarily focus on displaying dialogs and prompting for data.

USER INTERACTION OPTIONS

AppleScript developers frequently have the need to incorporate user interaction into their scripts. Sometimes, this interaction is simply to notify the user of a message or an error. At other times, there is a need to request input from the user. There are several options available to developers who need to incorporate user interaction into their AppleScript solutions.

For this particular article, we are going to stick to the basics. We will focus on the commands that make up the *User Interaction* suite in the *Standard Additions* scripting addition. These commands will allow your scripts to display basic dialogs and prompt users for common types of information. Please note that some of the code specified in this article will only function in *Mac OS X Panther (10.3)*, due to changes in the *Standard Additions* scripting addition terminology.

The *Standard Additions* scripting addition is installed by default with *Mac OS X*, and can be found in the *System > Library > Scripting Additions* folder.

For those looking to create more robust custom interfaces for their AppleScripts, there are other options available, including the following:

AppleScript Studio – This development environment, which is included on the *Xcode Developer Tools* CD that ships with *Mac OS X*, allows users to build interfaces for their AppleScript applications, giving them the look and feel of any other *Mac OS X* application. For more information about *AppleScript Studio*, visit - http://www.apple.com/applescript/studio.

FaceSpan – Similar in many respects to *AppleScript Studio*, this commercial application also allows users to create complete complex interfaces for their AppleScript applications. A key selling point for *FaceSpan* is it's extreme ease-of-use for novice users. For more information about *FaceSpan*, visit - http://www.facespan.com.

Smile – This free third-party script editing application allows users to create complex custom dialogs quickly and easily. For more information about *Smile*, visit - http://www.satimage.fr/software.

3rd Party Scripting Additions – Third-party scripting additions, such as *24U Appearance OSAX*, will allow users to dynamically build dialogs and interfaces for scripts during processing. For a comprehensive list of scripting additions, including those providing user interaction features, visit - http://www.osaxen.com.

The above listed tools range in complexity, and while some may be simple for beginners to master, others may be more complex and require more scripting experience. In the future, we will explore aspects of some of these other user interaction options.

ALERTS AND MESSAGES

Audio Alerts

Sometimes, you may need to provide input to a user without actually displaying a message on the screen. This can be done with an audio alert. Audio alerts are useful in scripts running on unattended machines, as they can easily attract attention from across the room. Audio alerts can also be useful to provide progress updates to the user during processing.

The *Standard Additions* scripting addition allows for two primary types of audio alerts - a beep, and a spoken message.

Audio alerts assume that the computer's sound level has been set to an appropriate level. However, this may not always be the case. You can use the **set volume** command, also available in the *Standard Additions* scripting

Benjamin Waldie is president of Automated Workflows, LLC, a firm specializing in AppleScript and workflow automation consulting. In addition to his role as a consultant, Benjamin is an evangelist of AppleScript, and can frequently be seen presenting at Macintosh User Groups, Seybold Seminars, and MacWorld. For additional information about Benjamin, please visit http://www.automatedworkflows.com, or email Benjamin at applescriptguru@mac.com.

addition, to change the volume level. Use this command to specify the desired volume level, from 0 (muted) to 7.

```
set volume 7
```

Please note that the *Standard Additions* scripting addition does not currently contain a command to GET the current volume level of the machine. Therefore, to get the volume level, you will need to utilize a third-party scripting addition, such as *Jon's Commands* http://www.seanet.com/~jonpugh/, or an application such as *Extra Suites* http://www.kanzu.com.

Beeps may be provided by using the **beep** command, and you may specify the desired number of beeps to occur with an integer. For example, the following code would beep 5 times:

```
beep 5
```

To provide a spoken message, use the **say** command. The following example shows how to use the **say** command, using the default voice assigned under *Speech* in *System Preferences*.

```
say "Hello!"
```

This code shows how you can specify which voice to use. Obviously, the voice specified must exist on your computer.

```
say "Hello!" using "Zarvox"
```

The **say** command also has another very interesting ability. It can actually be used to save a spoken message as a file, in *AIFF* format. For example, the following code will save the spoken message "Hello!" as an *AIFF* file, named "alert.aiff", to the user's desktop.

```
set theOutputFolder to path to desktop folder as string
set theOutputFile to theOutputFolder & "alert.aiff"
say "Hello!" saving to file theOutputFile
```

Display Dialog

Sometimes, it is necessary to provide more than an audio alert to a user during processing. You may want to provide a visual alert as well. In order to do this, you will need to make use of the **display dialog** command in the *Standard Additions* scripting addition.

```
display dialog  plain text
   [default answer  plain text]
   [buttons  a list of plain text]
   [default button  number or string]
   [with icon  number or string]
   [with icon  stop/note/caution]
   [giving up after  integer]
```

The **display dialog** command actually serves multiple purposes. It is used to display a message to the user, and it is used to get information back from the user, in the form of the button clicked, or the text that was entered.

The following code shows how to display a basic dialog to the user:

```
display dialog "Hello!"
```

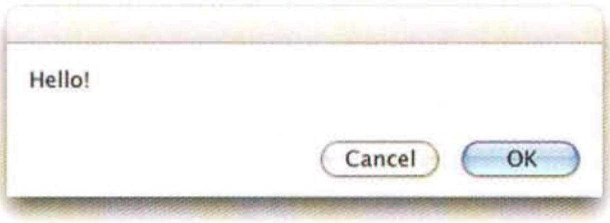


Figure 1. *A basic display dialog window*

The code above will display a dialog box containing the text "Hello!" along with a "Cancel" and an "OK" button. In some cases, it may be necessary to customize certain aspects of the dialog. For example, you may need to include more than two buttons, specify the names of the buttons, or include an icon.

The following code shows how to display a dialog containing custom buttons.

```
display dialog "How are you today?" buttons {"Lousy", "Good",
"Great!"}
```

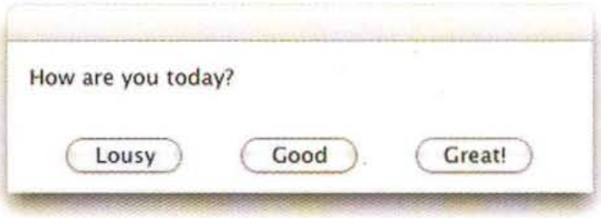


Figure 2. *A multi-button display dialog window*

You will notice, when running the code above, that the dialog does not contain a default button. If desired, you can specify which button should be used as the default button in the dialog.

```
display dialog "How are you today?" buttons {"Lousy", "Good",
"Great!"} default button 3
```

Or...

```
display dialog "How are you today?" buttons {"Lousy", "Good",
"Great!"} default button "Great!"
```

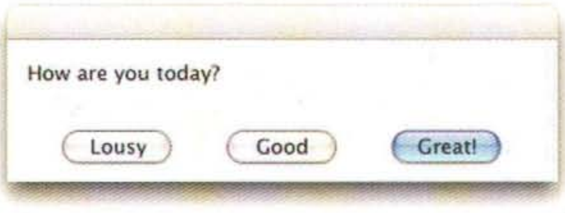


Figure 3. *A multi-button display dialog window with a default button*

A dialog can also be configured to display a *stop*, *note*, or *caution* icon. This can be done by specifying the icon's type, or its ID – **stop** (0), **note** (1), **caution** (2).

```
display dialog "How are you today?" buttons {"Lousy", "Good",
"Great!"} default button "Great!" with icon 1
```

Or...

```
display dialog "How are you today?" buttons {"Lousy", "Good",
"Great!"} default button "Great!" with icon note
```

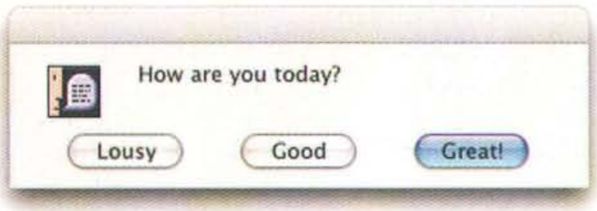


Figure 4. *A display dialog window with an icon*

In some cases, you may need your dialog to automatically dismiss. This is useful when displaying messages in scripts that must remain processing at all times, such as scripts running on unattended machines. The following code illustrates how to make your dialog automatically dismiss after a specified number of seconds.

```
display dialog "An error has occurred." giving up after 5
```

To prompt the user to enter text into your dialog, simply add the **default answer** parameter.

```
display dialog "Please enter a number:" default answer "5"
```

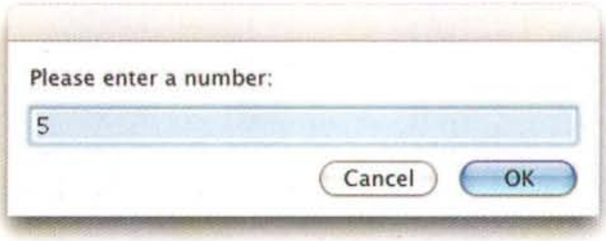


Figure 5. *A display dialog window with return text*

When displaying a dialog, you will generally want to have information returned to you for further processing. For example, you may need to determine which button the user clicked, or what text the user entered, and then take an appropriate course of action. Regardless of the type of dialog displayed, the **display dialog** command will always return a value. This value will indicate, in the form of an AppleScript record, the text that was entered (if relevant), which button was clicked, and whether the dialog was automatically dismissed (if relevant).

```
{text returned:"5", button returned:"OK", gave up:false}
```

By adding repeat loops, try statements, etc., you can begin to create more complex dialogs that will check the results entered by the user. For example, the following code will prompt the user to enter a number, and will keep re-displaying the prompt until a number is entered.

```
set thePrefix to ""
set theNumber to ""
set theIcon to note
repeat
  display dialog thePrefix & "Please enter a number:" default
answer theNumber with icon theIcon
  set theNumber to text returned of result
  try
    if theNumber = "" then error
    set theNumber to theNumber as number
    exit repeat
  on error
    set thePrefix to "INVALID ENTRY! "
    set theIcon to stop
  end try
end repeat
display dialog "Thank you for entering the number " &
theNumber & "."
```

Prompts

The *Standard Additions* scripting addition also contains several other user interaction commands, which will allow you to prompt a user for various types of specific information.

Selecting Files or Folders

In some cases, you may need your script to prompt the user to select one or more files or folders. This may be done to determine which files to process, or to select an input or output folder. To prompt the user to select a file, use the **choose file** command. To prompt the user to select a folder, use the **choose folder** command.

For folders, you can optionally specify a prompt, a default location, whether or not the dialog should allow a user to select invisible folders, and whether the user should be allowed to make multiple selections.

```
choose folder
  [with prompt  plain text]
  [default location  alias]
  [invisibles  boolean]
  [multiple selections allowed  boolean]
```

For files, you have the option to specify generally the same information you can specify for folders, along with a list of acceptable file types, if desired. By specifying a list of file types, you can limit the files that the user may select.

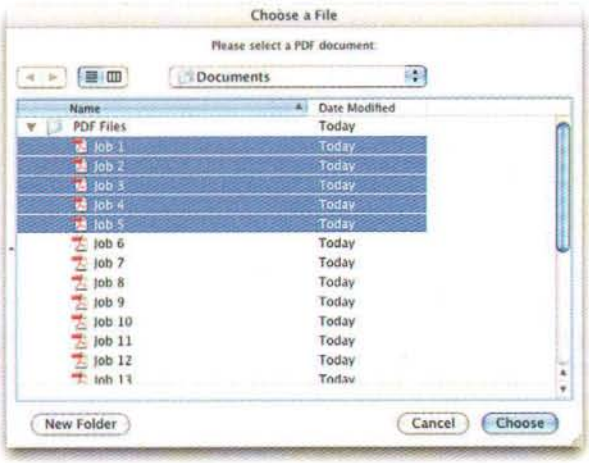
```
choose file
  [with prompt  plain text]
  [of type  a list of plain text]
  [default location  alias]
  [invisibles  boolean]
  [multiple selections allowed  boolean]
```

It is important to note that both the choose folder and choose file commands are set to not allow for multiple selections by default. In addition, the choose file command is set to display invisible files by default, and the choose folder command is set to not display invisible folders by default. Therefore, you will need to add the appropriate optional parameters if the default behavior is not desired.

```
choose file without invisibles

choose folder with multiple selections allowed
```

The following code will prompt the user to select one or more PDF files, using the *Documents* folder as the default directory.

```
choose file with prompt "Please select a PDF document:" of
type {"PDF "} default location (path to documents folder) with
multiple selections allowed
```



Name	Date Modified
PDF Files	Today
Job 1	Today
Job 2	Today
Job 3	Today
Job 4	Today
Job 5	Today
Job 6	Today
Job 7	Today
Job 8	Today
Job 9	Today
Job 10	Today
Job 11	Today
Job 12	Today
Job 13	Today

Figure 6. *A choose file prompt*

Both the choose file and choose folder commands will return either an alias, or a list of aliases (if multiple selections were allowed). For example:

```
alias "Macintosh HD:Users:bwaldie:Documents: PDF Files:Job
1.pdf"
```

Or…

```
{alias "Macintosh HD:Users:bwaldie:Documents: PDF Files:Job
1.pdf", alias "Macintosh HD:Users:bwaldie:Documents: PDF
Files:Job 2.pdf"}
```

Prompting for a File Name

The choose file name command in the *Standard Additions* scripting addition may be used to prompt the user to enter a name, and specify a location for a file. This can be useful if you need your script to create or save a document in a user-specified location.

New

PrimeBase 4.2 Replication Server

Check out the fully programmable

Replication Server

- Bidirectional Updates supported
- Update 3rd-party DBMS
- Send Emails
- Post/get Data to/from Websites

[SQL-Runtime Plugin for REALbasic $ 499,-]

All PrimeBase Server Software

- SQL-Runtime Libraries available
- SQL Database Server
- Application Server
- Replication Server
- Open Server

Available on the most popular platforms

- Completely cross-platform
- Full-text searching and indexing
- Mac OS & OS X
- Linux
- Solaris
- IBM AIX
- all Windows platforms

PrimeBase

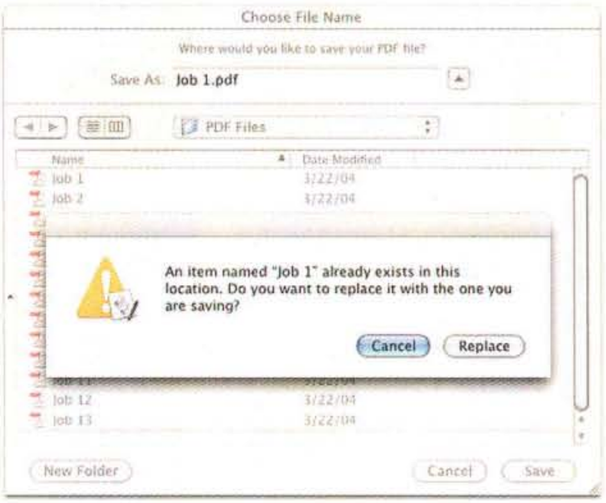
SNAP Innovation GmbH
Altonaer Poststraße 9a
D-22767 Hamburg / Germany
www.primebase.com
e-mail: info@primebase.com
Fon: ++49 (40) 389 044-0
Fax: ++49 (40) 389 044-44

```
choose file name
   [with prompt  plain text]
   [default name  plain text]
   [default location  alias]
```

The prompt that will be displayed as a result of the choose file name command will even handle the task of asking the user whether to replace an existing item with the same name. However, please keep in mind that the prompt will not actually overwrite or delete the existing file. You must write AppleScript code to perform this function.

The choose file name command will allow you to optionally specify a prompt, a default file name, and a default location.

```
choose file name with prompt "Where would you like to save
your PDF file?" default name "Job 1.pdf"
```



Where would you like to save your PDF file?

Job 1.pdf

```
Job 1     3/22/04
Job 2     3/22/04
Job 12    3/22/04
Job 13    3/22/04
```

An item named "Job 1" already exists in this location. Do you want to replace it with the one you are saving?

Figure 7. *A choose file name prompt*

The result of the choose file name command will be a file reference.

```
file "Macintosh HD:Users:bwaldie:Documents: PDF Files:Job
1.pdf"
```

Selecting Items in a List

One of the more common tasks that you may need to perform, is to have the user make a selection from a list. This can be done using the choose from list command.

```
choose from list  a list of plain text
   [with prompt  plain text]
   [default items  a list of plain text]
   [OK button name  plain text]
   [cancel button name  plain text]
   [multiple selections allowed  boolean]
   [empty selection allowed  boolean]
```

The choose from list command will allow you to optionally specify a prompt, the item(s) that should be selected by default, and the OK and Cancel button names. You may also optionally indicate whether the user should be allowed to make multiple selections, or make an empty selection.

The following code will prompt the user to select a favorite type of fruit:

```
choose from list {"Apples", "Oranges", "Peaches"} with prompt
"Please select your favorite type of fruit:" default items
{"Apples"}
```

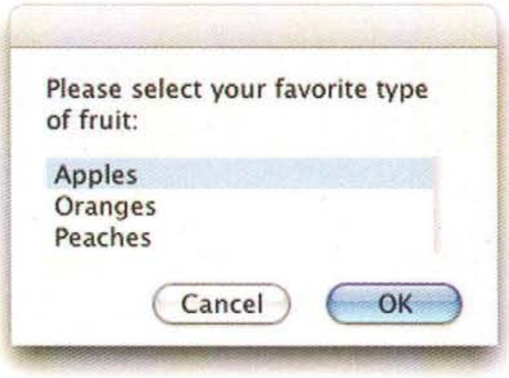


Figure 8. *A choose from list window*

If the user makes a selection, the choose from list command will return the user's selection(s), in list format.

```
{"Apples"}
```

Please note that while most dialogs and prompts will return a user interaction error (error number –128) if the user clicks the *Cancel* button, the choose from list command will return a value of false instead.

Selecting an Application

The *User Interaction* suite in the *Standard Additions* scripting addition also contains a command that will allow you to prompt a user to select an application – choose application.

```
choose application
   [with title  plain text]
   [with prompt  plain text]
   [multiple selections allowed  boolean]
   [as  type class]
```

The choose application command will allow you to optionally specify a window title, a prompt, and whether the user should be able to select multiple applications.

By default, the choose application command will return a reference to the chosen application. However, you may optionally specify that the command should return an alias to the application instead. In addition, if the user is allowed to select multiple applications, the result of this command will be a list.

```
choose application

--> application "Mail"
```

```
choose application as alias

--> alias "Macintosh HD:Applications:Mail.app:"

choose application with multiple selections allowed

--> {application "iChat", application "Mail"}

choose application as alias with multiple selections allowed

--> {alias "Macintosh HD:Applications:iChat.app:", alias
"Macintosh HD:Applications:Mail.app:"}
```

Selecting a Color

A command that is new to *Mac OS X Panther (10.3)* is the choose color command. Invoking this command will display the standard *Mac OS X* color picker palette. Once the user has made a selection, it will be returned as a list of RGB values, i.e. {0, 9820, 65535}.

You may optionally specify a default color to display.

```
choose color
   [default color  RGB color]
```

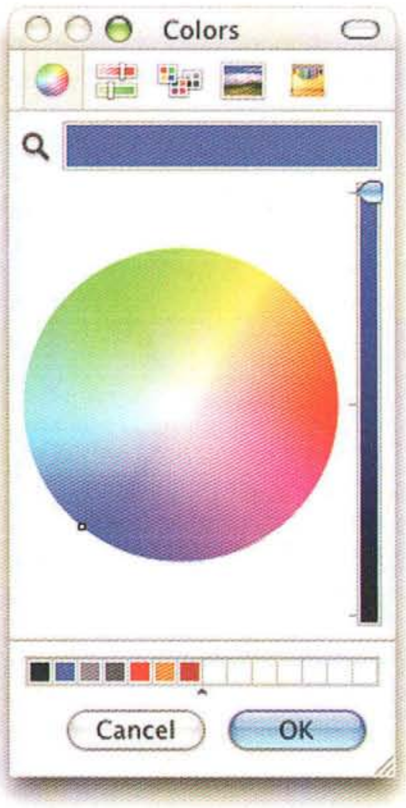


Figure 9. *A choose color dialog*

Prompting for a URL

The choose url command may be used to prompt the user to select a network service, such as a server. This command will allow you to optionally specify which network services to display to the user, and whether or not the user should be allowed to manually enter a URL. Please note that this command does not actually connect to a chosen network service. Instead, it will simply return the user's selection as a URL string.

```
choose URL
   [showing  a list of Web servers/FTP Servers/Telnet
hosts/File servers/News servers/Directory services/Media
servers/Remote applications]
   [editable URL  boolean]
```

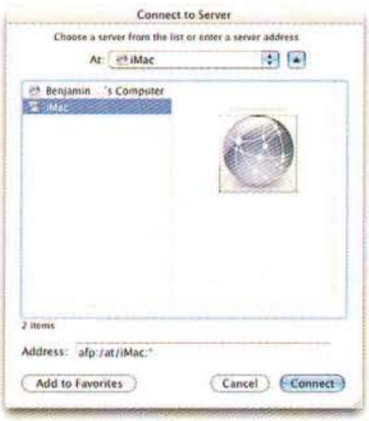


Figure 10. *A choose url dialog*

In Closing

One final command that I would like to mention is the delay command. While I am not sure that I would consider this to be a user interaction command, it has been placed in the *User Interaction* suite of the *Standard Additions* scripting addition. It is also a very useful command, and is certainly worth mentioning. The delay command can be used to make your script pause for a desired number of seconds. For example, the following code would pause the script for 60 seconds.

```
delay 60
```

Hopefully, this in-depth look at the commands in the *User Interaction* suite of the *Standard Additions* scripting addition will encourage you to begin adding more user interaction into your scripts. By adding this functionality into your AppleScripts, you will not only make your scripts feel more like "real" applications, but you will also begin to eliminate those hard-coded folder and file paths, names, and more. This will also help to make your script more portable, and reduce the possibility of errors.

In next month's article, we will start looking at some other basic AppleScript functionality, and, in the future, we will try to touch on some of the other, more robust user interaction options. Until next time, keep scripting!

By Dave Wooldridge

Communicating with Customers

How the New CAN-SPAM Act Affects Software Developers

Over the last few months, you've undoubtedly heard talk of the new CAN-SPAM Act that became federal law in the United States on January 1, 2004. The purpose of the CAN-SPAM Act is to help regulate and reduce the onslaught of spam that plagues all of our inboxes, but the perceived effectiveness of the new law has been met with mixed reactions. Many people feel that the law is not strict enough and actually legitimizes unsolicited e-mails. Still receiving hundreds of spam messages per day, touting organ enlargement solutions, get rich quick pyramid schemes, generic prescription painkillers and Viagra alternatives? You're not alone.

Whether you believe in the validity of the CAN-SPAM Act is irrelevant. Violations carry a severe penalty. Depending on the specific offense, penalties include jail sentences of one to five years, plus hefty monetary fines. CAN-SPAM also discusses the offering of a bounty of at least 20% of the collected fines to citizens who report violators. And with the strong hatred the general public has for spam, you can count on seeing a lot of people submitting reports, especially if they can make money for putting spammers behind bars! As a federal law in the U.S., it supercedes any regional state anti-spam legislation that currently exists, although some states may still pursue their own prosecution of spam offenders. Needless to say, anti-spam laws should not be taken lightly. To read the entire CAN-SPAM Act in its entirety, visit http://www.spamlaws.com/federal/108s877.html

For those of you who find legal documents to read more like alien Klingon scripture than actual English can rest easy. This month's column will summarize the key points of the CAN-SPAM Act and how they directly affect software developers. As a programmer communicating with only your existing customers and opt-in e-newsletter subscribers, you may wonder why you should be concerned with this new law, especially since many of the rules apply to unsolicited e-mail. The simple truth is that people lead busy lives and while they may have voluntarily subscribed to your e-newsletter last October, come January they may not have any recollection of doing so when they receive your latest e-mail. They report it as spam, not remembering they actually did subscribe at some point, and then before you know it, you've got men in black suits with badges knocking on your front door. You can never be too careful or conscientious when it comes to people's e-mail privacy. Even if you employ double opt-in procedures for your e-newsletter and customer mailing lists, it is important for you to understand the law. Honor the new rules with every bulk e-mail campaign you deliver, so that you do not accidentally wind up on the wrong side of a lawsuit. Plus, for those of you who do maintain a regular e-newsletter mailing list (and if you don't, you should), we'll explore some effective techniques for maintaining your subscriber base, strengthening customer loyalty and increasing software sales.

The "From" Address

The "From" line should state your name or your business name with a valid e-mail address. Never forge a false e-mail address and never substitute the name with marketing jargon (like "Special Offers").

Besides the fact that it is against the law, using a false or different e-mail address other than the one you are sending from is a sure-fire way to get caught by spam filters. For example, you decide to send the latest e-newsletter to your mailing list over the weekend from home, using your personal DSL account, but you want the e-mail to look professional, so you forge the "From" line to be your company e-mail address. While this may seem harmless enough, two problems can arise. One, your ISP's mail server may detect the false "From" header and block the bulk e-mail from being sent. And two, many of the anti-spam filters that people use are smart enough to spot false "From" headers and automatically send your e-newsletter to the "Junk Mail" folder where it gets deleted. You don't want to spend valuable time and money producing an e-newsletter that never gets read.

Dave Wooldridge is the founder of Electric Butterfly (`www.ebutterfly.com`), the web design and software company responsible for HelpLogic, Stimulus, UniHelp, and the popular developer site, `RBGarage.com`.

The "Reply-To" Address

Always use a "Reply-To" e-mail address that is guaranteed to be valid and active for at least thirty (30) days after you send the e-mail.

For those recipients who decide they want to unsubscribe, they often do not scroll down far enough to see the unsubscribe instructions at the bottom of the message, so they will click the Reply button and e-mail you a request to be removed from the list. Manually processing these kinds of unsubscribe requests can be tedious and time-consuming, so an effective way to deal with this problem is to set-up the "Reply-To" e-mail address with an auto-responder message. Most web hosting providers and ISPs support an e-mail auto-responder option, which shoots back a pre-defined message to any person who e-mails that address. Most companies use them for notifying clients when specific employees are on vacation, etc. but they can also be used to provide customer service to your mailing list subscribers. Set up a new e-mail address such as replies@yourcompany.com that will not be used for any other purpose than as your e-newsletter's "Reply-To" address. Add an auto-responder message to that e-mail account. The message should thank the sender for contacting your company and explain politely that in order to unsubscribe from the mailing list, they should visit the following link (and then include the unsubscribe URL). It's also a good idea to list the URL or e-mail address for contacting customer support in case the sender wanted to ask a question or receive technical assistance on one of your software products. Make sure your auto-responder is in plain text format to ensure that even text-only e-mail programs can properly display the message.

The "Subject" Line

Your e-mail's "Subject" line should be descriptive and directly related to the content of your message. Using misleading or deceptive "Subject" lines (such as "Your account has been declined!" for a home loans promotion) can carry hefty penalties.

Using vague "Subject" lines may seem like a brilliant ploy to get curious recipients to open your e-mail message, but in today's world of sophisticated spam filters and over-cautious users, e-mails such as "Check this out!" or "Deal of the Century!" will, more often than not, end up unread in the "Junk Mail" folder. So not only are they against the law, but these questionable "Subject" lines will not stand out in an already oversaturated sea of vague spam headlines.

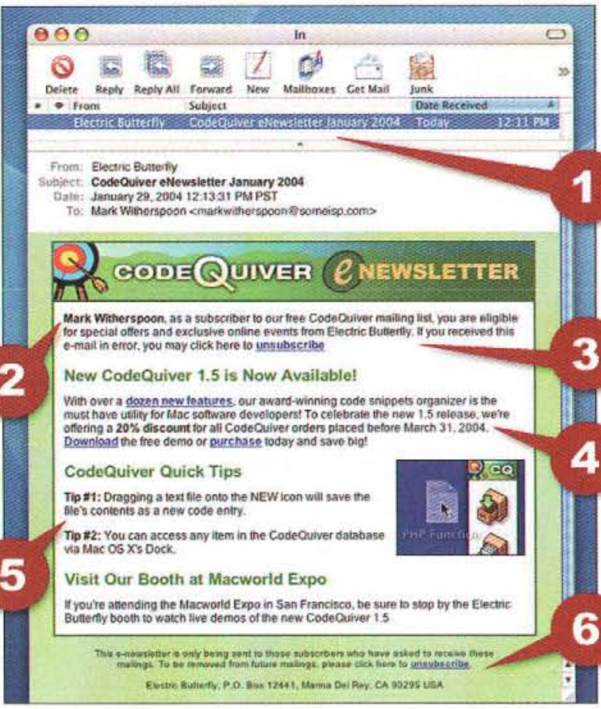
The best code editor on the planet is now available for the Mac.

slickedit.

Develop an edge.

For more information, visit www.slickedit.com/mac 1.800.934.EDIT +1 919.473.0070

SlickEdit is a registered trademark of SlickEdit Inc. Other names may be trademarks of their respective owners.



Electric Butterfly CodeQuiver eNewsletter January 2004 Today 12:11 PM

From: Electric Butterfly
Subject: **CodeQuiver eNewsletter January 2004**
Date: January 29, 2004 12:13:31 PM PST
To: Mark Witherspoon <markwitherspoon@someisp.com>

Mark Witherspoon, as a subscriber to our free CodeQuiver mailing list, you are eligible for special offers and exclusive online events from Electric Butterfly. If you received this e-mail in error, you may click here to unsubscribe

New CodeQuiver 1.5 is Now Available!

With over a dozen new features, our award-winning code snippets organizer is the must have utility for Mac software developers! To celebrate the new 1.5 release, we're offering a **20% discount** for all CodeQuiver orders placed before March 31, 2004. Download the free demo or purchase today and save big!

CodeQuiver Quick Tips

Tip #1: Dragging a text file onto the NEW icon will save the file's contents as a new code entry.

Tip #2: You can access any item in the CodeQuiver database via Mac OS X's Dock.

Visit Our Booth at Macworld Expo

If you're attending the Macworld Expo in San Francisco, be sure to stop by the Electric Butterfly booth to watch live demos of the new CodeQuiver 1.5

This e-newsletter is only being sent to those subscribers who have asked to receive these mailings. To be removed from future mailings, please click here to unsubscribe.

Electric Butterfly, P.O. Box 12441, Marina Del Rey, CA 90295 USA

Figure 1. *Even though recipients voluntarily subscribed to your opt-in mailing list, take extra care to ensure that your e-mail messages are effective, while remaining compliant with state and federal laws.*

Your goal is to create a subject line that catches the attention of recipients while honoring the true content of your e-mail message. Including your company name in the "Subject" line may prove effective, but unless you're a well-known entity like Adobe or Macromedia, people may not instantly recognize your name. For small software companies and independent shareware developers, the eye-catching word that will jog users' memories is most likely the name of your product. Using our fictional software product, CodeQuiver, as an example, the "Subject" line in **Figure 1, Item 1** is "CodeQuiver eNewsletter January 2004." You could obviously make the line longer with additional words such as "Electric Butterfly January 2004 eNewsletter: CodeQuiver Update," but you run the risk that users' have their incoming e-mails listed with a narrow width for the "Subject" field, displaying only the first part of the phrase: "Electric Butterfly January 2004 eNewslet..." By placing the identifying word "CodeQuiver" in the beginning of the phrase, "CodeQuiver" will always be visible, no matter how narrow the "Subject" field is set in email programs. Our "Subject" line will now prove to be much more eye-catching for those users who quickly recognize the product name, but not our company name.

But what if your e-newsletter promotes more than one software product? Due to space limitations, you don't want to list all your products in the "Subject" line. If you have several software titles and they cater to different markets (some are consumer-oriented, some are developer tools, etc.) then you may want to consider sending out several e-newsletters with each one targeted to specific subscribers in your mailing list. Users of your consumer-based products may not want to read in-depth news about your developer tool offerings. The most effective e-newsletters are those that are brief and focused on the interests of the recipient. Less is more. They will remain subscribers as long as the content remains useful and is easy to read quickly.

Apple is a good example of a company that utilizes e-newsletter targeting. When signing up online to receive e-mail news from Apple, you are asked to select the specific e-newsletters that interest you (usually represented by a list of checkboxes). "Apple eNews" announces the latest updates, special offers and user tips for Mac OS X, iLife, and other consumer-based Apple products. "QuickTime News" showcases multimedia authoring, special promotions on Final Cut Pro, Final Cut Express, and other QuickTime-based software. "New Music Tuesdays" provides the latest music news and releases available in Apple's iTunes Music Store. "ADC News" is the official e-newsletter for Apple Developer Connection, focusing on Xcode, Cocoa, and third-party developer tools. If Apple combined all of these different topics into one consolidated newsletter, it would weaken their effectiveness to generate sales. There would be too much information in one e-mail with much of the content reaching the wrong people whose interests lie in other areas. Let's say a proud father who creates iMovie videos of his daughter's soccer games is interested only in hearing about the latest iLife and QuickTime news. If he received a general, "all-in-one" Apple e-newsletter that was packed with lots of unrelated articles on Objective-C programming techniques, he may grow impatient wading through the content just to find the information that suits his needs. If he unsubscribes after a few issues, Apple then loses the ability to tell him about new products and special offers that he *would* be interested in, losing a potential sale. By subscribing to the targeted e-newsletters "Apple eNews" and "QuickTime News," he only receives news related to the topics he wants to read about, providing a consistent and reliable line of communication from Apple to his e-mail inbox.

If you don't have a way to configure and manage targeted groups within a single mailing list on your own web server, there are several third-party services such as Microsoft bCentral's ListBuilder (http://www.listbuilder.com/) which provide these advanced features and more. ListBuilder generates a sign-up web form that enables subscribers to select their topics of interest (which ListBuilder calls "Groups") from a list that you pre-define. ListBuilder hosts your mailing list database and automatically manages all submissions and unsubscribe requests. Using ListBuilder's convenient web admin tools, you can elect to send an e-newsletter to either the entire list or to only specific Groups.

E-Mail Identifiers

If you e-mail an unsolicited message promoting your products or services, you must identify the e-mail as an advertisement in the "Subject" line. Although there is no predefined standard, most e-marketers use the label [ADV]. Make sure the label is easy to spot by placing it at the beginning of the "Subject" field. For example, "CodeQuiver Improves Programming Productivity!" would become "[ADV] CodeQuiver Improves Programming Productivity!"

Solicited e-mails that are delivered to opt-in subscribers are not required to include any kind of identifier, but since some recipients may accidentally forget that they did subscribe voluntarily, you can avoid false spam accusations by politely reminding them in the first paragraph of your message. Personalize the e-mail with the recipient's name (**Figure 1, Item 2**) to show that it's not a generic message, but that it's addressed specifically to them. Then add a brief sentence that reminds them that they *chose* to receive your e-mails (**Figure 1, Item 3**). This can be done in a very elegant way by mixing the phrase with some dialog that explains why it is beneficial for them to *continue* receiving the e-mail messages. Using the fictional CodeQuiver e-newsletter as an example, we accomplished both goals with one sentence by saying: "As a subscriber to our free CodeQuiver mailing list, you are eligible for special offers and exclusive online events from Electric Butterfly."

Any e-mails (both solicited and unsolicited) that include sexually oriented material must be labeled as such in the "Subject" line. Most software developers won't have a need for this kind of labeling, but just in case some of you develop adult multimedia applications or web sites, it might be helpful to note that many adult magazines use the identifier [NUDITY] in the "Subject" line when e-mailing advertisements to their subscribers. Some states in the U.S. consider the unsolicited delivery of sexually oriented e-mails to be against the law and enforce serious penalties, so it is strongly recommended to consult an attorney before sending e-mails with this kind of content.

Unsubscribe Requests

Whether you are sending solicited or unsolicited e-mails, each and every e-mail is required to include an easy to find "unsubscribe" link somewhere in the body of the message. A recipient should be able to opt-out from your mailing list in one step (or no more than two steps). Your opt-out instructions should be very easy to follow, making the process extremely simple. Forcing recipients to jump through a complicated string of steps as a deterrent to keep them from unsubscribing is not only against the law, but will only serve to agitate potential customers. All unsubscribe requests must be honored within ten (10) days of receiving them.

Most e-marketers place the "unsubscribe" link at the bottom of the e-mail message. For opt-in e-newsletters, this is a good place to once again remind recipients why they are receiving the e-mail. In **Figure 1, Item 6**, the "unsubscribe" link is accompanied by a brief sentence of instructions: "This e-newsletter is only being sent to those subscribers who have asked to receive these mailings. To be removed from future mailings, click here to unsubscribe". Just in case some recipients do not think to scroll down to the bottom of the message, placing a second "unsubscribe" link near the top of the message provides added convenience. A subtle way to do this without cluttering your content is shown in **Figure 1, Item 3**, where the "unsubscribe" link is also added to the tail-end of the introductory paragraph. This may seem like overkill, but if you only include the one "unsubscribe" link at the very bottom, you'll be surprised by how many e-mails you receive from people who cannot figure out how to opt-out.

Another way to prevent confusion is to provide a confirmation page that informs people that their unsubscribe request was successful and that they should not receive any more mailings from you. If your mailing list system requires subscribers to opt-out by replying via e-mail with the word "unsubscribe" in the "Subject" line, use an auto-responder to send an e-mail message back to the user, acknowledging their request.

It is your responsibility to ensure your automated opt-out process is working properly. Double and even triple-check your opt-out feature. If people are unable to successfully unsubscribe from your mailing list, their first guess will not be a faulty system. They will assume the worst: that you are trying to take advantage of their good will. Feeling frustrated and exploited, these potential customers just turned into angry enemies who may falsely report you as a spammer.

A Physical Postal Address

All solicited and unsolicited e-mails are required to include the valid postal address of the sender. This means your full street address, city, state/province, zip code, and country. It's standard practice to place the postal address at the very bottom of the e-mail message, beneath the unsubscribe instructions.

If you are a home-based shareware developer and would rather not give out your home address (for fear of customers knocking on your front door on a Saturday afternoon), then rent a P.O. box from your local post office or mail supplies store. Prices are very affordable, typically ranging from $40 to $100 (US) per year. Using a P.O. box as your public company address will also shield your home residence from receiving junk mail from address collectors.

Guilty By Association

The CAN-SPAM Act also bans automated e-mail harvesting, so programming a robot script to crawl web sites to collect e-mail addresses is strictly prohibited. This means that purchasing a list of harvested e-mail addresses from a third-party company or utilizing a bulk e-mail service whose database consists of harvested e-mail addresses is also against the law.

Not only are you responsible for your own e-mail practices, but you are also responsible for all affiliates, resellers, and marketing agencies who send out e-mails on your behalf. Whether you have directly instructed these partners to promote your products/services or if they are acting independently in the hopes of increasing their own commissions, you are ultimately responsible for their actions since the e-mails are tied directly to the promotion of your software company. This is usually an easy element to control with resellers and marketing firms, but it can become staggeringly difficult to manage with an affiliate program. Enticing web site owners to promote your software by using special purchase URLs that track and reward commissions based on sales they generate can be an extremely powerful marketing tool. Affiliate programs like this have proven to be incredibly successful for online retailers like Amazon.com, but managing the selling tactics of these affiliate members can be somewhat tricky. Since affiliates make money when customers buy your software through their special affiliate hyperlinks, they often barrage the public with web advertisements and unsolicited e-mail promotions in the hopes of increasing their commissions. To enforce that all affiliates obey local and federal spam laws, all program members should be required to sign a Terms & Conditions document that limits your liability and states their selling boundaries. Let it be known in no uncertain terms that any affiliates who violate the rules will have their membership terminated and may face legal repercussions.

EXTRA MEASURES

Beyond the new CAN-SPAM Act, there are several things you can do to further safeguard your e-newsletter campaigns and make them more effective as a key communication platform between you and your subscribers.

Double Opt-in. Don't let people clutter your mailing list with invalid e-mail addresses or the unauthorized e-mail addresses of others. After signing up online, a verification message should be sent to the submitted e-mail address, asking the owner to confirm their subscription request by clicking the included URL. If the request was initially made in error, they can opt-out by simply doing nothing. If the submitted e-mail address is invalid, then it won't be added to the mailing list (since there is no one on the receiving end to confirm the request). Making the opt-in path a simple two-step process will ensure that your subscribers are genuinely interested in receiving your e-newsletters.

Privacy Policy. Every web site and mailing list should have a privacy policy. Many people are so afraid of ending up on unsolicited spam lists that they refuse to submit their e-mail address in any online form unless you tell them upfront that their e-mail address will never be sold or distributed to any one outside of your company. On the same screen as your e-news web form, include a brief sentence guaranteeing that you will never share their personal information and include a URL link to your privacy policy. Some e-marketers even include the "privacy policy" URL at the bottom of every e-mail (in the same paragraph as the "unsubscribe" link).

Never Use Attachments. With so many computer viruses being distributed through e-mail, the general public has become very wary of e-mail attachments. The fear has escalated to the point where e-mails with attachments that were not sent by a friend or family member are usually quickly deleted without being opened. If you're sending an HTML e-newsletter, never send the HTML document as an attachment – always embed the HTML into the main body of the e-mail. And never send the web graphics used in the HTML e-newsletter as attachments. Aside from the fear factor of attachments, using relative paths to attached graphics in your <IMG> tags does not work properly in some e-mail programs, incorrectly displaying your HTML e-newsletter with broken images. The professional way to display graphics within your HTML e-mails is to host the graphics on your web server and then use absolute URLs in your <IMG> tags. For example, instead of using a relative path:

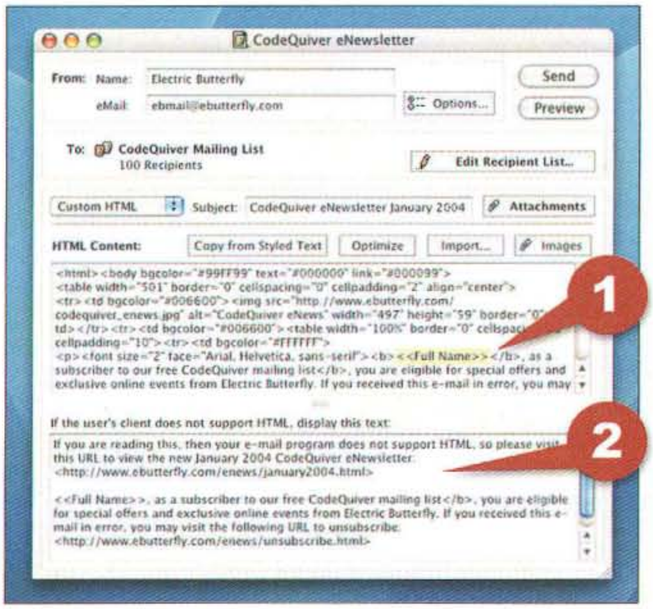
```
<img src="logo.jpg" width="300" height="50" border="0">
```

use an absolute path:

```
<img src="http://www.yoursite.com/logo.jpg" width="300" height="50" border="0">
```

Never Use HTML Forms. Avoid embedding forms in your HTML e-mails. Besides the fact that some older e-mail programs do not support HTML forms, they tend to scare or intimidate recipients. People are often suspicious of what is being activated when the "Submit" button is clicked and where their data is being sent. Even if your subscribers trust your company, they may not trust the e-mail since they have no way to prove its authenticity. The last couple years have seen hundreds of e-mail scams disguising themselves as eBay or PayPal, asking customers to verify their passwords, bank account numbers, social security numbers, etc. through e-mail-based forms. Not wanting to be the next victim, subscribers may opt to simply delete the e-mail or worse yet, unsubscribe. If you want e-mail recipients to participate in an online poll or survey, post the HTML form on your web site and then promote it in your e-mail with a URL link to that form page.

Useful Information. Don't assume people will remember what your software does by only promoting the latest release or sale price in your e-newsletters. In **Figure 1, Item 4**, we promote the new CodeQuiver release while briefly reminding readers of CodeQuiver's primary function. And if your e-newsletters only encourage software purchases, then their usefulness may diminish in the eyes of readers. By providing convenient user tips and techniques (**Figure 1, Item 5**), recipients will be motivated to remain an active subscriber for a much longer period of time.



Electric Butterfly
ebmail@ebutterfly.com

CodeQuiver Mailing List
100 Recipients

Custom HTML

CodeQuiver eNewsletter January 2004

```
<html><body bgcolor="#99FF99" text="#000000" link="#000099">
<table width="501" border="0" cellspacing="0" cellpadding="2" align="center">
<tr><td bgcolor="#006600"><img src="http://www.ebutterfly.com/
codequiver_enews.jpg" alt="CodeQuiver eNews" width="497" height="59" border="0"
td></tr><tr><td bgcolor="#006600"><table width="100%" border="0" cellspac
cellpadding="10"><tr><td bgcolor="#FFFFFF">
<p><font size="2" face="Arial, Helvetica, sans-serif"><b><<Full Name>></b>, as a
subscriber to our free CodeQuiver mailing list</b>, you are eligible for special offers and
exclusive online events from Electric Butterfly. If you received this e-mail in error, you may
```

If the user's client does not support HTML, display this text:

```
If you are reading this, then your e-mail program does not support HTML, so please visit
this URL to view the new January 2004 CodeQuiver eNewsletter:
<http://www.ebutterfly.com/enews/january2004.html>

<<Full Name>>, as a subscriber to our free CodeQuiver mailing list</b>, you are eligible
for special offers and exclusive online events from Electric Butterfly. If you received this e-
mail in error, you may visit the following URL to unsubscribe:
<http://www.ebutterfly.com/enews/unsubscribe.html>
```

Figure 2. *Using an e-mail merge program like IntelliMerge, you can send personalized e-mails to your customer mailing list. Sending your e-mail with a multi-part MIME type will allow your message to display on both HTML-enabled and text-only e-mail programs.*

Personalize. If you decide not to use a mailing list service such as Microsoft bCentral's ListBuilder or custom web server software, but elect to e-mail your mailing list from your own computer, you should still personalize each e-mail with individual names and e-mail addresses (**Figure 1, Item 2**). Personalizing your e-newsletters and customer communications adds a nice touch, building a direct bridge between you and each recipient. Do not send one message with your entire mailing list hidden in the BCC field. Many ISPs will refuse delivery, auto-detecting these kinds of bulk e-mails as spam, and even if they do get past ISP gateways, most personal spam filters will discard them. The best way to send out your own e-newsletter is by sending it to each recipient individually as a separate, personalized e-mail. Manually, this would take far too much time, but there are several mail merge software applications available for the Mac that can help automate this process. Intelli Innovations' IntelliMerge (http://www.intellisw.com/intellimerge/) is one such solution. Import your mailing list into IntelliMerge and then embed special database tags in your e-newsletter's text to dynamically populate those tags with the appropriate subscriber information when e-mailed via IntelliMerge. For example, the <<Full Name>> tag (**Figure 2, Item 1**) will be replaced with the subscriber's name when IntelliMerge delivers the e-mail (**Figure 1, Item 2**). If you prefer to use Mac OS X's Mail and Address Book, then Christian Fries' SerialMail (http://www.christian-fries.de/osx/SerialMail/) may be a viable solution for you.

Catering to Text-Only E-mail Programs. Due to personal preferences, limited access to the latest e-mail software, and/or the ongoing threat of e-mail-driven viruses, some people are unable to read HTML e-mails or choose to disable the HTML option. Since these precious few may be either existing customers or potential new customers, don't exclude them from enjoying your e-newsletters. This doesn't mean you have to design text-only e-mails to meet the lowest common denominator. You can configure the e-mail with a multi-part MIME type. This enables one part of the message to be designated as HTML, while the other part is relegated to plain text. Setting the current message in IntelliMerge as "Custom HTML" turns it into a "multi-part" e-mail, displaying two fields: one for the HTML version of your e-newsletter and one for the plain text version. In HTML-compatible e-mail programs, the HTML version will automatically display, leaving the plain text version hidden. In text-only e-mail programs, the plain text version will be displayed by default. If you prefer everyone to read the more elegantly designed HTML version, you can post it as a web page on your site and include the URL and viewing instructions in the plain text section (**Figure 2, Item 2**), so that text-only recipients can view the HTML e-newsletter in their web browsers.

STAY INFORMED

Before sending out bulk e-mails from your own computer or web server, review your ISP or web hosting provider's e-mail policies. E-mailing thousands or even hundreds of people at one time can set off "red flags" with your ISP and web hosting provider. You don't want to send out your latest software announcement and then have your account shut down for violating an existing e-mail policy. Internet access providers are becoming increasingly strict in an attempt to control and reduce spam. Although you may only be delivering an e-newsletter to your double opt-in list of happy, legitimate subscribers, your ISP is not aware of these details – all they see are traffic logs of 3,000 e-mails being sent from your account within the short span of a few minutes. Using a third-party service like Microsoft bCentral's ListBuilder avoids these kinds of complications.

Stay on top of the latest local and federal anti-spam developments. Even though your opt-in e-newsletter mailing list would not be considered spam, it's a good idea to be aware of any new changes or amendments, so that your e-mails are always compliant with the law. When in doubt, consult an attorney so as to choose the right e-mail strategy that best protects you and your company. Better safe than sorry in the precarious world of e-mail marketing.

Cool Things Happen

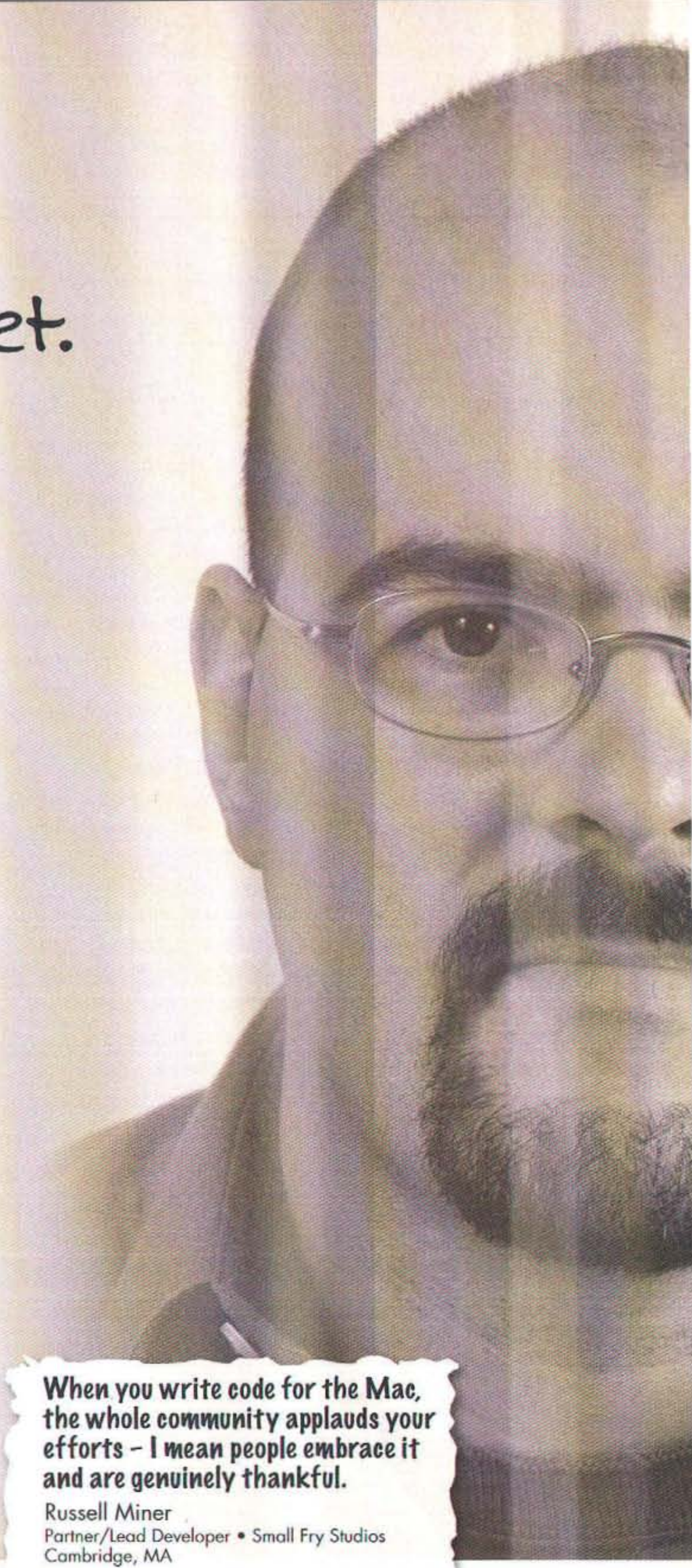
Conference July 12-15, 2004
Expo July 13-15, 2004

Boston, MA
Boston Convention & Exhibition Center

Be part of the revolution at
www.macworldexpo.com

Macworld
Conference & Expo®

When Mac Users Meet.



The Revolution Continues in Boston!

Connect with like-minded people and exchange your ideas, success stories and experiences at Macworld this July. Experience breakthrough products and services from leading companies devoted to you and the Mac OS platform.

We have the most innovative and knowledgeable industry experts leading our world-renowned conference programs and activities. Learn great things from the best!

Pro Conference: for programmers, power users, administrators, integrators and other Mac OS masters!

Users Conference: for educators, designers, musicians, "switchers" and new users!

Power Tools Conferences: in-depth training on iLife projects, FileMaker, home recording studios and other popular topics!

MacLabs: hands-on workshops on DVD Studio Pro, Acrobat/PDF, color management and other key topics!

Get Connected: Geeks & Gadgets demonstrations, MacBrainiac Challenge, Birds-of-a-Feather meetings and much more!

Discover powerful technology, make connections, and see for yourself why Macworld Conference & Expo® is the #1 event for the Mac community.

When you write code for the Mac, the whole community applauds your efforts – I mean people embrace it and are genuinely thankful.

Russell Miner
Partner/Lead Developer • Small Fry Studios
Cambridge, MA

Register online with Priority Code: D13Ø2

IDG WORLD EXPO

Flagship Sponsors:

Macworld Macworld.com MacCentral

By Dan Shafer

Runtime Revolution: Programming for Mere Mortals

This program has its roots in HyperCard, but its reach is stratospheric (First in a series of four articles)

Who Cares?

I can hear you now. "Oh, goodie. Yet another programming language. Just what we need. What do I need with another language? Isn't [insert your personal favorite *language du jour here*] good enough?" Leaving aside for the moment the question of why any serious software developer would confine him or herself to a single programming language, let's just say that if you want to develop Mac Classic and OS X applications that also happen to run virtually unchanged on Windows and *nix platforms, Revolution may just be the sweetest thing that's happened to you in a long, long time.

If you're a seasoned professional programmer who writes only for the Macintosh, you may still find Revolution a useful tool. You can use it to prototype user interfaces quickly, turn those prototypes into demos for users to evaluate as you rapidly modify the UI in response to user feedback.

If you're one of perhaps hundreds of thousands of people I call Inventive Users, you're going to love the ease of programming and application design in Revolution as well as the way it gets along with corporate databases, the Web, and rich media. (Inventive Users are people who don't program for a living but who are willing to take time to learn an accessible scripting language and use it to create applications they or their departments need or want.)

What Is Revolution?

Revolution, a product of Edinburgh, Scotland-based Runtime Revolution Ltd., grows out of HyperCard, a much-beloved and widely used Apple Computer product that the company abandoned a few years ago, much to the chagrin of tens of thousands of its customers, including some of the biggest companies in the world. HyperCard ran only on Macintoshes and was a strictly black-and-white application for most of its life. When color *was* added, near the end of the HyperCard life cycle, it was not well thought out and never worked very well.

Scott Raney loved HyperCard but he needed something like it to run on Unix. So he set about to develop a HyperCard-like product for his platform of choice and wound up creating the MetaCard engine, about 10 years ago. Kevin Miller and his Scottish cohorts licensed the MetaCard engine, put a more robust and attractive programming UI on it and extended it in some other crucial ways, marketing the result as Revolution. Recently, Runtime Revolution acquired MetaCard and Raney now works for the engine's new owners.

Revolution combines a powerful, graphical IDE (integrated development environment) with a highly accessible programming language called Transcript to create a platform for developing a broad range of software applications that will run virtually unchanged on Mac Classic, OS X, every flavor of Windows currently supported by Microsoft (and some that aren't) and many versions of *nix, including the widely popular Linux. Applications are in color, take full advantage of native UI widgets as much as possible, and offer performance comparable to Java and Basic applications.

In fact, I like to call Revolution Java without the Java. Revolution delivers on the promise of true cross-platform development better than Java in my experience, and does so at a fraction of the programming time and effort.

A Quick Tour

Over the next four months, I'll give you a pretty good handle on Revolution and the Transcript programming language. In this starter piece, I'll walk through an overview of both the IDE and Transcript, just to give you a flavor for it. By the time we're done with the series, you'll be able to develop fairly complex — or at least interesting and useful — applications on your own.

(You can download a 30-day free trial version of Revolution at the company's Web site at http://www.runrev.com. That version will be perfectly suited for this set of articles, but, of course, it

Dan Shafer is a freelance writer and software developer who wrote some of the most widely read books about HyperCard and other Macintosh programming technologies over the past 15 years. He is the author of more than 60 books on computing and high technology. His most recent title is *Revolution: Software at the Speed of Thought*, published by Revolution Press and designated the product's official third-party book. The first of the three-volume set was released at MacWorld Expo in San Francisco earlier this year. He also hosts RevolutionPros.com, a members-only Web site for Revolution users. You can reach him at dan@shafermedia.com.

will expire before all the articles have run. Revolution offers an inexpensive version of the product, however, that you can use to complete your tour.)

Walking Through the IDE

When you launch Revolution, you will see a desktop much like the one shown in **Figure 1**.

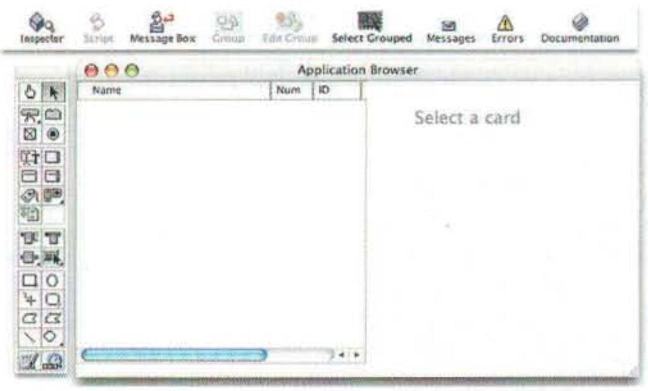


Figure 1. *Default opening Revolution desktop showing standard IDE*

I've rearranged the elements of the screen from the way they may appear on your system, to accommodate a smaller figure. You can see three of the most important tools in the Revolution IDE:

the **toolbar**, which appears at the top of the screen in a fixed location and houses buttons to access commonly needed tools in the IDE

the **tools palette**, a floating window shown on the left of the screen in Figure 1, from which you choose tools to lay out windows and create graphics

the **Application Browser**, which can be resized and relocated and which acts as a way to navigate within and inspect the objects in a Revolution application or window

There are several other tools in the IDE but two are particularly worth a few moments' time at the outset of your learning experience. One is the message window, which you can cause to appear by typing Command-M, selecting the Message Box tool from the toolbar, or selecting the Message Box item from the **Tools** menu. The other interesting tool is the extensive online documentation in Revolution.

The Message Box is an interactive programming window. Any Transcript code you type into the top portion of the Message Box will execute exactly as it will in your finished application. I love programming environments that provide this kind of support; in fact, I do not enjoy coding in development tools that don't allow me to try out code quickly and interactively.

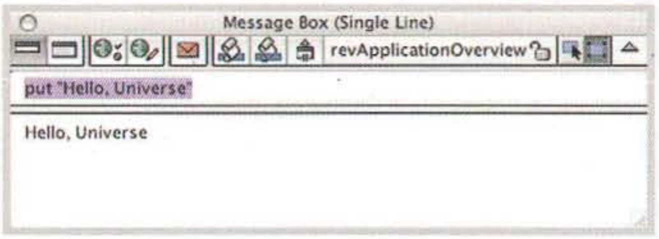
Open the Message Box and type `put "Hello, Universe"` into the top portion. Press Return. Your result should look something like **Figure 2.**

Revolution at a Glance for HyperCard Scripters

If you have experience developing Mac "stackware" using HyperCard, you'll be right at home in Revolution and Transcript. But you'll feel like your "home" got a huge upgrade. Here are some of the key features in Revolution that HyperCard developers will welcome with open arms, it not joyous weeping.

- True, native color
- High-performance script execution that removes the need for most XCMD and XFCN resources you needed with HyperCard
- Cross-platform delivery; you can create applications on your beloved Mac and sell them to people who are still stuck with Windows. Linux is supported as well.
- External database access. While you can still store data in your Revolution stacks (with a slight twist in design), you can also interact directly with SQL databases, information stored in external files, and the entire Internet.
- Great support for Internet protocols makes it possible for you to build Web-aware software without mastering the intricacies of the protocols themselves.
- A scripting language that contains about five times as many commands, keywords, and other language elements as HyperTalk, yet retains the approachable syntax and much of the forgiveness of HyperTalk.
- Conversion of existing HyperCard stacks. With a little preparation, almost any HyperCard stack that doesn't use window-management externals can be ported to run in Revolution.
- Virtually every HyperTalk command works as you expect it to in Transcript. There's even support for some language elements that are no longer relevant so that those commands won't break if you use them in new or imported stacks.

There's a lot more. The Revolution folks have thoughtfully provided a special document that delineates in great detail the similarities and differences for HyperCard and SuperCard developers. I highly recommend reading those documents before you attempt any work in Revolution; it'll save you a lot of fumbling around with things that almost work as you expect, but not quite.

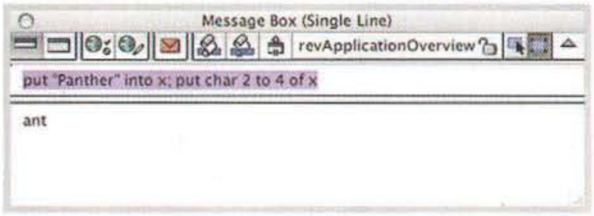


```
put "Hello, Universe"
```

```
Hello, Universe
```

Figure 2. *Revolution's Message Box is an interactive programming window*

The put command places its argument into the bottom portion of the Message Box, even when the command is executed from inside a Revolution script. You can type multiple commands into the Message Box and the result, if you create one, will still appear in the bottom pane of the Message Box, as you can see in **Figure 3.** You just have to separate commands with semicolons.



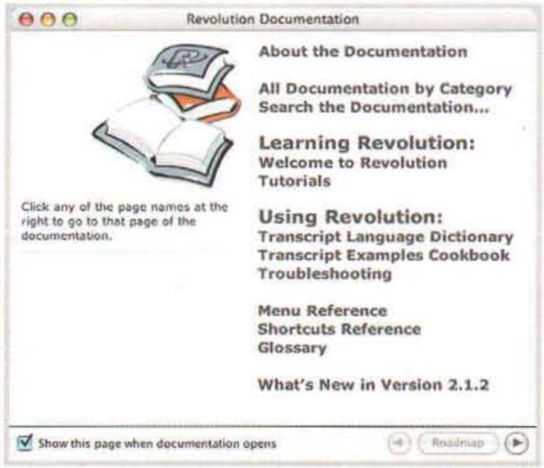
```
put "Panther" into x; put char 2 to 4 of x
```

```
ant
```

Figure 3. *Multiple lines of code in Message Box*

The Message Box is actually fairly powerful in its own right but I don't want to take time to go into all of its features here or we'll run out of room for the rest of this overview.

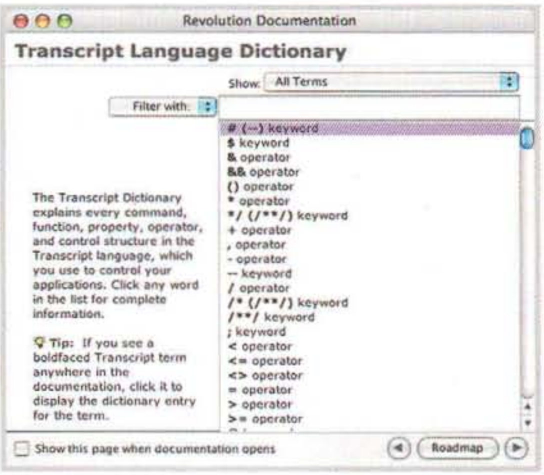
Online documentation in Revolution is better than that in any programming environment I've worked with in a long time. Click on the Documentation icon in the Tool Bar and you'll see the interactive table of contents for the online documentation, shown in **Figure 4**.


Revolution Documentation
About the Documentation
All Documentation by Category
Search the Documentation...
Learning Revolution:
Welcome to Revolution
Tutorials
Using Revolution:
Transcript Language Dictionary
Transcript Examples Cookbook
Troubleshooting
Menu Reference
Shortcuts Reference
Glossary
What's New in Version 2.1.2
Show this page when documentation opens
Roadmap

Click any of the page names at the right to go to that page of the documentation.

Figure 4. *Table of Contents for online Revolution documentation*

As you can see, the online documentation includes a full-text search capability, a product overview, several tutorials, a complete dictionary of Transcript, a cookbook of ready-made code examples, and a number of other useful elements. Click on the Transcript Language Dictionary option and you'll see a window something like **Figure 5**.

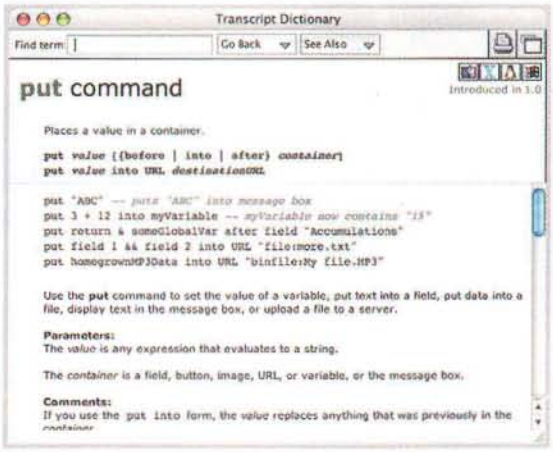


The Transcript Dictionary explains every command, function, property, operator, and control structure in the Transcript language, which you use to control your applications. Click any word in the list for complete information.

Tip: If you see a boldfaced Transcript term anywhere in the documentation, click it to display the dictionary entry for the term.

Figure 5. *Opening page of Transcript Language Dictionary*

Selecting a Transcript language element from the scrolling list opens another window with complete documentation (see **Figure 6**) including syntax, examples, and detailed explanations of when and how to use the element and what to expect when you do. Each such window also has a "See Also" menu which is often quite useful in helping you locate the precise command or other language element you need.



put command

Places a value in a container.

```
put value [{before | into | after} container]
put value into URL destinationURL
```

```
put "ABC" -- puts "ABC" into message box
put 3 + 12 into myVariable -- myVariable now contains "15"
put return & someGlobalVar after field "Accumulations"
put field 1 && field 2 into URL "file:more.txt"
put homegrownMP3Data into URL "binfile:My file.MP3"
```

Use the **put** command to set the value of a variable, put text into a field, put data into a file, display text in the message box, or upload a file to a server.

Parameters:
The *value* is any expression that evaluates to a string.

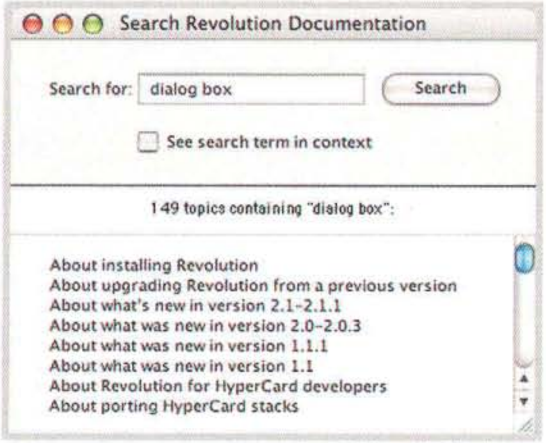
The *container* is a field, button, image, URL, or variable, or the message box.

Comments:
If you use the put into form, the value replaces anything that was previously in the

Figure 6. *Sample documentation page*

Figure 6 shows the full syntax for the put command, provides five examples of its use, tells you when and where to use it, defines its parameters, and offers comments that describe details of its use in certain situations. Using the printer icon in the upper right corner of this window prints a nicely formatted set of pages containing the documentation.

Let's take a quick look at how to use the interactive documentation to learn to do something a bit more conventional for our first Transcript program. Click the "Roadmap" button near the bottom right corner of the open documentation window. This takes you back to the table of contents. Now click on the Search the Documentation link. We want to put the "Hello, Universe" message in a dialog box rather than having it appear in the Message Box. Type "dialog box" (without the quotation marks) into the search box and click on the Search button. The result looks like **Figure 7**.

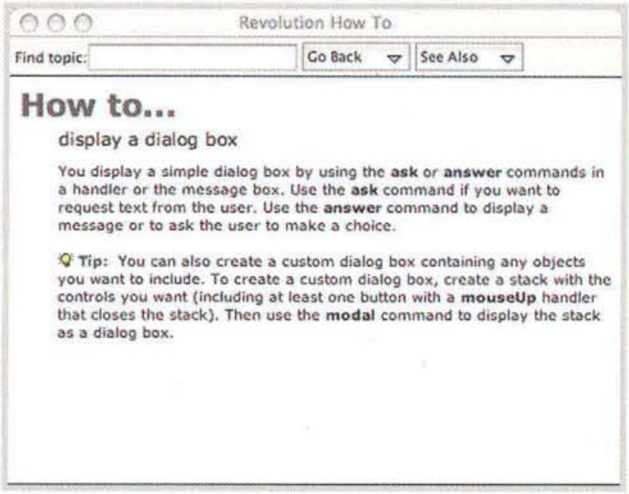


149 topics containing "dialog box":

About installing Revolution
About upgrading Revolution from a previous version
About what's new in version 2.1-2.1.1
About what was new in version 2.0-2.0.3
About what was new in version 1.1.1
About what was new in version 1.1
About Revolution for HyperCard developers
About porting HyperCard stacks

Figure 7. *Documentation search for "dialog box"*

We want to learn how to do something, so let's scroll down to where the "How To" items are sorted together. Sure enough, there's an entry called "How to display a dialog box." Click on that entry and you see the documentation shown in **Figure 8**.



How to...

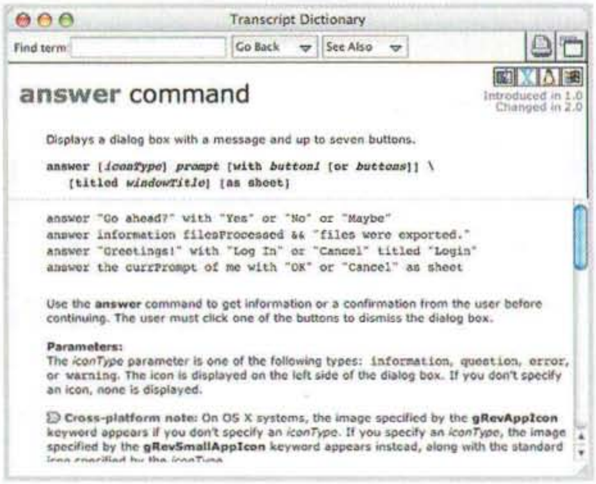
display a dialog box

You display a simple dialog box by using the **ask** or **answer** commands in a handler or the message box. Use the **ask** command if you want to request text from the user. Use the **answer** command to display a message or to ask the user to make a choice.

Tip: You can also create a custom dialog box containing any objects you want to include. To create a custom dialog box, create a stack with the controls you want (including at least one button with a **mouseUp** handler that closes the stack). Then use the **modal** command to display the stack as a dialog box.

Figure 8. *Documentation describing how to display a dialog box*

A quick read of the first paragraph tells us we want to use the answer command because we're not asking the user for any text when we display our greeting message. Now, click on the word "answer" in the documentation (the fact that the term is bolded means it's a link to further documentation). Sure enough, up pops the Transcript dictionary entry for the answer command, shown in **Figure 9**.



answer command

Displays a dialog box with a message and up to seven buttons.

```
answer [iconType] prompt [with button1 [or buttons]] \
   [titled windowTitle] [as sheet]
```

```
answer "Go ahead?" with "Yes" or "No" or "Maybe"
answer information filesProcessed && "files were exported."
answer "Greetings!" with "Log In" or "Cancel" titled "Login"
answer the currPrompt of me with "OK" or "Cancel" as sheet
```

Use the **answer** command to get information or a confirmation from the user before continuing. The user must click one of the buttons to dismiss the dialog box.

Parameters:
The *iconType* parameter is one of the following types: information, question, error, or warning. The icon is displayed on the left side of the dialog box. If you don't specify an icon, none is displayed.

Cross-platform note: On OS X systems, the image specified by the **gRevAppIcon** keyword appears if you don't specify an *iconType*. If you specify an *iconType*, the image specified by the **gRevSmallAppIcon** keyword appears instead, along with the standard icon specified by the *iconType*

***Figure 9.** Documentation for* answer *command*

Reading the comments, we find that if we don't supply any button names after a with key word, Revolution supplies the user with a simple "OK" button. That's exactly what we want, so let's try using the Message Box again. Enter the command answer "Hello, Universe!" and press Return. The result is shown in **Figure 10.**

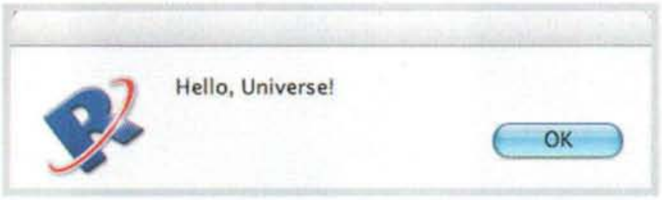


***Figure 10.** An* answer *dialog box created from the Message Box*

(You can control the icon displayed in the dialog box, but we don't have time to go into that in this article.)

The Tools Palette and Window Layout

One of the coolest features in Revolution — which is not dissimilar in this respect to other graphical IDEs — is the ability to lay out and design windows using simply drag-and-drop or select-and-click operations starting in the Tools Palette.

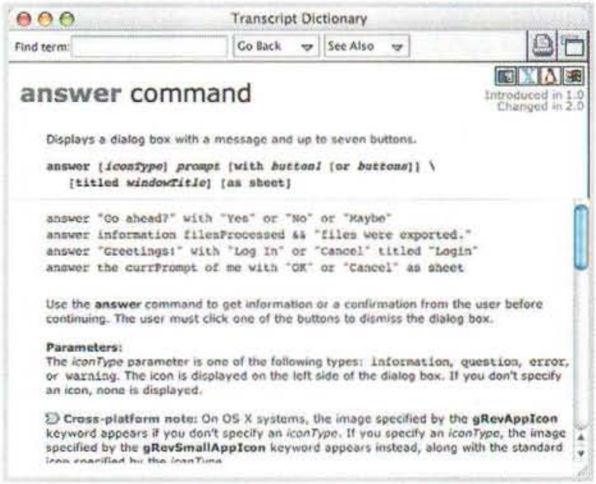
The Tools Palette includes tools for the quick and easy creation of the following UI elements:

- four types of buttons (standard push buttons, tabbed buttons, radio buttons, and checkbox buttons)
- five types of fields (regular, scrolling, list, scrolling list, and uneditable labels)
- scrollbars (vertical and horizontal)
- report objects (for compiling and generating printed reports)
- three types of menus (popup, pulldown and option/combo-box)
- menu items
- seven basic graphical shapes as well as polygons
- image objects
- QuickTime player objects

This does not, by the way, exhaust the types of UI components Revolution supports. The **New Control** submenu of the **Object** menu lists a number of specialized versions of some of these controls.

To use these controls, you need a window in which to work. In Revolution, you create the first window in each application by selecting **New Mainstack** from the **File** menu. A new, blank window appears. Now you can click on an object in the Tools Palette and then click in the window to create an object of the chosen type using its default size. There are other ways of creating new components in the window, but we'll stay with this one for this article.

Figure 11 shows a window with a single button and a single field. You can create this, if you're following along, simply by clicking on the button tool in the Tools Palette and then clicking in the window. Drag the newly created button to its desired location, then do the same with a field.



answer command

Displays a dialog box with a message and up to seven buttons.

```
answer [iconType] prompt [with button1 [or buttons]] \
   [titled windowTitle] [as sheet]
```

```
answer "Go ahead?" with "Yes" or "No" or "Maybe"
answer information filesProcessed && "files were exported."
answer "Greetings!" with "Log In" or "Cancel" titled "Login"
answer the currPrompt of me with "OK" or "Cancel" as sheet
```

Use the **answer** command to get information or a confirmation from the user before continuing. The user must click one of the buttons to dismiss the dialog box.

Parameters:
The *iconType* parameter is one of the following types: information, question, error, or warning. The icon is displayed on the left side of the dialog box. If you don't specify an icon, none is displayed.

Cross-platform note: On OS X systems, the image specified by the **gRevAppIcon** keyword appears if you don't specify an *iconType*. If you specify an *iconType*, the image specified by the **gRevSmallAppIcon** keyword appears instead, along with the standard icon specified by the *iconType*

***Figure 11.** Sample window with one button and one field*

Quick Overview of the Transcript Language

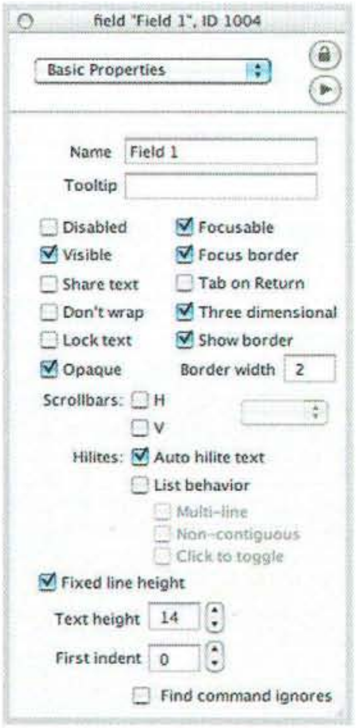
Now that we have a window with two objects in it and we know a little about how to display a dialog box with information for the user, let's take a very quick peek at the Transcript language and turn our little window into an application that has at least some marginal utility.

Programming in Transcript consists of creating scripts that are associated with objects. Scripts, in turn, contain one or more handlers, each of which is designed to respond to a particular message sent around the Revolution system. A handler always begins with the keyword on, followed by the name of the message to which the handler is designed to respond. For example, if you want to write a script with a handler that does something when the user clicks the mouse, you write a handler that starts with the words on mouseUp, since mouseUp is the name of the Revolution message that gets sent when the user releases the mouse over the same object he pressed the mouse. The handler must close with the keyword end followed by the message name, in this case end mouseUp. Between those two lines you put the code you want Revolution to execute when the object whose script you are writing receives the message.

That's really almost all there is to the Revolution programming model. There is a hierarchy of objects through which messages travel so that you can decide at what level to intercept and respond to a message, and you can create your own custom messages with corresponding handlers, but those are refinements of the basic idea.

OK, back to our little window. We know how to use an answer dialog to display some hard-coded text to the user. Let's shift the design just a bit and make the process more dynamic. We'll design this little application so that, whatever the user types into the field, gets displayed in an answer dialog box.

First, we need to know the name of the field so we can tell Transcript where to get the contents to display in the answer dialog. Select the field and double-click on it. That opens its Property Inspector (see **Figure 12**), which is where you can control virtually every property or aspect of the field's appearance. The information we want, the name of the field, is right there for us to read. It's called "Field 1."


field "Field 1", ID 1004

Basic Properties

Name: Field 1
Tooltip:

☐ Disabled ✓ Focusable
✓ Visible ✓ Focus border
☐ Share text ☐ Tab on Return
☐ Don't wrap ✓ Three dimensional
☐ Lock text ✓ Show border
✓ Opaque Border width 2

Scrollbars: ☐ H ☐ V

Hilites: ✓ Auto hilite text
☐ List behavior
☐ Multi-line
☐ Non-contiguous
☐ Click to toggle

✓ Fixed line height

Text height 14
First indent 0

☐ Find command ignores

Figure 12. *Property Inspector for field*

Now we just need to write a handler in the button's script that will display the contents of the field when the mouse is clicked on the button. To do so, we have to open a script editor for the button. There are a number of ways to do this, but for now, just select the button and then hold down the Option and Command keys simultaneously. The editing window opens, ready for us to write a handler in the button's script.

Just as a guess, let's try this handler:

```
on mouseUp
  answer field 1
end mouseUp
```

In the script editing window, when you type on mouseUp, notice that Transcript supplies the handler-ending statement and positions your cursor correctly for the first line of executable code.

When you're done, hit the Apply button and close the script editor.

Now, switch from the selection tool to the browse tool (the hand in the upper left of the Tools Palette), type some text into the field, and press the button. The result should look something like **Figure 13.**

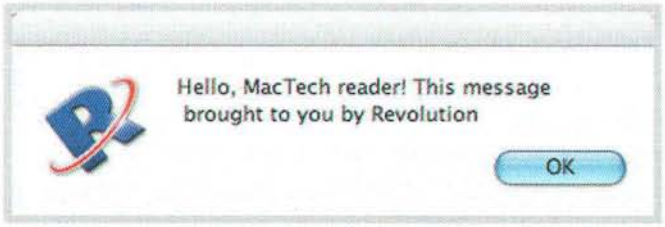

Hello, MacTech reader! This message brought to you by Revolution

Figure 13. *Dialog box containing text of field in sample window*

You can insert carriage returns into the text in Field 1 and the dialog box will format it exactly as you enter it.

Many objects you create in Revolution applications will have more than one handler in their scripts. The Transcript editor provides you with tools to focus on one handler at a time, see all the handlers in a script, navigate directly to the one you want, and several other helpful tools that make scripting even fairly complex objects simple and approachable.

By John C. Welch

The Orange Micro iBot Pro

No, really, the iSight isn't the only game in town

THE IBOT

iSight, iSight, iSight. If you listen to the iHype, you'd iThink that the all good FireWire cams must start with "I". Now, the iSight is an excellent webcam, but it's hardly the only one out there. Orange Micro's iBot was available long before the iSight, and in many cases, is a better choice.

Basics

The iBot is the embodiment of the "uniSight". It's got a great bulbous body, a stand that sort of resembles a foot, and the FireWire cable is permanently attached to the body. It looks like a blue and white translucent tennis ball that grew an eye, a foot and a tail. There's no mounting clips included, since you don't mount it. You set it down on a flat surface.

That's the first problem with the iBot. Since the cable comes straight out of the back of the iBot, it can sometimes be a pain to set up correctly. The cable is on the thick side, so depending on how it runs from the iBot to the computer; you may have to resort to devices like books, or duct tape to get the iBot to stay in one position. The cable attachment also limits vertical orientation, since the weight of the cable can tip the iBot if you tilt the camera too far down. On the plus side, the base attaches to the sides of the iBot, and while the iBot can pivot on those points, it's hardly loose, so once you get it tilted correctly, it's not going to change that orientation because of a loose pivot. If you can deal with cable and mount issues, the iBot gives you a far greater vertical adjustment range than the iSight's mount allows for. A thinner, detachable cable would be a definite plus in any case. Allowing for different kinds of mounts would be a plus too.

Using the iBot

Even allowing for its differences, the iBot is quite easy to use. You have to manually focus, which can be tedious, but once the focus is set, you can leave it alone. (There are times when you just want to set the cam up and have it stay that way. The iSight's auto focus, while mostly convenient, can be quite annoying if there's a lot of motion in the cam's field of vision. Anything the iSight works with, the iBot should work with just fine. I found picture quality between the iSight and the iBot to be comparable, and while the iBot doesn't include a microphone, is just as good a cam as the iSight.

With recent upgrades to iMovie allowing you to use cams such as the iBot as video sources, you can now use the iBot as a very cheap way to get video into your Mac. (No, you are NOT going to get the same quality from an iBot, as you will from a real DV camera. But as a quick 'n' dirty workaround, it's not bad. The iBot comes with an excellent array of software, especially in the Pro configuration, which was the review config. Among the choices is the iVisit video conferencing package, and BTVPro, an excellent basic DV capture utility that has excellent AppleScript support. The iBot pro also comes with an analog headset mic, which makes video conferencing a lot more convenient, and is a nice touch. (Now, if Orange Micro were to eventually replace that with their Bluetooth headset that would be a very sweet package indeed.) Considering the iBot Pro package is about $30 less than the iSight, yet includes a lot more features and software, it's a compelling alternative to the iSight, albeit with an unfortunate base design.

CONCLUSION

If you decide on the iBot Pro, you shouldn't worry that you won't be one of the iKool iKids. The package is a great deal, and it's built around a great cam.

John Welch <jwelch@provar.com> is an IT Staff Member for Kansas City Life Insurance, a Technical Strategist for Provar, (http://www.provar.com/) and the Chief Know-It-All for TackyShirt, (http://www.tackyshirt.com/). He has over fifteen years of experience at making Macs work with other computer systems. John specializes in figuring out ways in which to make the Mac do what nobody thinks it can, showing that the Mac is a superior administrative platform, and teaching others how to use it in interesting, if sometimes frightening ways. He also does things that don't involve computertry on occasion, or at least that's the rumor.

By Jono Bacon

Qt

A multi-platform graphical toolkit

The birth of the toolkit

In the software development world, there are many tools and services available to help developers maximise their application development cycle, and squeeze every bit of functionality out of their products. These tools include compilers, debuggers, development environments, profilers, and, of course, graphical toolkits. In this article, I will review a toolkit that is rapidly growing in reputation and ability; Qt, the flagship product from Norwegian company Trolltech. They state that Qt can make the "Code Less. Build More. Compile Anywhere" motto a reality.

In this article I will be reviewing the latest version of Qt available at the time of writing, version 3.2.2. I will assume you have never used Qt before, and so will evaluate the different aspects of the Qt system. Although I will be discussing many different features, not all of them are specific to the new 3.2.2 version; I will discuss specific features in this new version towards the end of the article. This way those of you familiar with Qt can find out what is new in 3.2.2, and those not can get a perspective on the whole Qt toolkit, and its features.

It's all about the code

Qt is a multi-platform toolkit. "Nothing new there" I hear you scream, citing Java as an example, but Qt offers a slightly different method of creating graphical applications. First, Qt is a native toolkit. This means that when you create a Qt application and compile it, a native binary is created to run on each specific operating system. The speed implications for the application are therefore drastically improved, and large applications should work, theoretically, as fast as any other native software, with the added benefits of Qt's cross platform source code.

The actual cross platform nature of Qt is largely a feature of its Application Programming Interface (API). The concept here is that the source code remains the same for each platform version of Qt, so to create a binary for another platform, you just recompile using the Qt for that platform. Sounds great in theory, but does it work? We will investigate this later in the article. Before we get onto the cross platform nature of Qt, however, let us first look at what Qt can do and what is available in it.

Qt Features

I have been familiar with Qt as a product since the release of the first version, and each major release has put more and more functionality under the hood. The first and most basic set of functionality is for creating graphical interfaces. Qt includes widgets (the equivalent of Controls) for buttons, checkboxes, radio buttons, tabs, icon panes, canvases, dialog boxes, and other common interface elements. In addition to these elements, there are special dialog boxes (such as a file picker, about box, etc.) and facilities that save the developer from having to re-implement these functions over and over.

In addition to these graphical components, Qt includes a number of additional features. One of the most critical set of features are the convenience classes for handling data and types. Qt provides a number of these classes to support Arrays, Strings, Vectors, Maps, and many more types. In addition to these basic classes there are also classes to handle networking, sockets, file transfer, sound, and many other aspects of software development. It is good to see that Trolltech has created a rich API that not only provides graphical widgets, but also provides a number classes that make the day to day tasks of programming it a little bit easier. Another feature in Qt is the concept of additional modules. These extra modules have specific functionality that can be added to the API. These modules include a graphical canvas, database access, networking, OpenGL, and XML facilities.

Although Qt is primarily a graphical toolkit and convenience classes, Trolltech has worked to supplement this API with tools and facilities to assist in the development of projects. These additional tools include the graphical dialog box creation tool Qt Designer, the translation tool Qt Linguist, the documentation tool Qt Assistant, and the compilation tool QMake. We will look at each of these tools later in this review.

Qt Usage

Qt is a C++ based toolkit, and Object Orientated Programming (OOP) is fundamental to using Qt. Each of the components is available as a class (such as a QPushButton to create a push button widget), and each class has a number of methods to handle common tasks and features. Although some developers use procedural programming for graphical applications, the nature of OOP lends itself well to GUI programming due to the fact that inheritance is

Jono Bacon is a writer, musician, and developer based in the United Kingdom. Jono has an avid interest in Linux/Open Source and has been involved with a number of Open Source projects.

fundamental to GUI's. As an example, there is the general concept of a button (something that you click on), and then specific types of buttons (push, radio, toolbar, etc). In Qt, inheritance is used in this way so that the general QButton class is inherited by a more specific QPushButton class, for example. This process gives the developer all the functionality that could be needed for that specific widget, and the lower level functionality for its inherited class. This is incredibly flexible and is implemented well in Qt.

Development of applications can be quite varied in Qt. When developers begin programming with a toolkit, they often use classes with a lot of functionality that is rarely used, that does nothing but increase the size of the binary. In Qt things are a little different. There are different classes that serve different uses. As an example, there is a QMainWindow class that provides a lot of functionality for typical office type applications with menus, toolbars, and a main content area. Although great for this kind of functionality, it may seem a little bit of overkill for more simplistic applications or graphical elements. Say you wanted to create a simple window with a single text box to type your password into, you could use a QDialog class that is much more efficient.

One of the most interesting features of Qt is the way in which user interaction is handled. In many toolkits there is the concept of events, messages, call backs, etc., where a particular widget will start a particular event when the user does something. It is then the programmers responsibility to capture the event and do something constructive in response to it. Trolltech has taken this (often bizarre and complicated) concept and refined it, coming up with a solution named Signals and Slots. The basic idea is that each class has a number of pre-defined signals that are emitted when something happens, such as clicking on a button or selecting an item in a menu.

For example, the QPushButton class that is used to create a simple push button (such as an OK and Cancel button in a dialog box) has a clicked() signal; this signal is emitted when someone clicks on the button. This particular signal can then be connected to a slot, which is any normal function. Using this system, you can easily connect any function to a user interaction. It requires only a single line of code to perform this connection.

Additional Tools

Earlier I mentioned that there are some tools included with Qt that can assist in developing your applications. These tools come in the form of Qt Designer, Linguist, Assistant, and QMake. All of these tools are genuinely useful, and speed up development with Qt.

Qt Designer

Qt Designer essentially gives you the ability to draw your graphical interface visually by dropping interface elements onto a window. Qt Designer is a very flexible tool, and has support for all of the graphical widgets that are available in the toolkit. Not only can you add items such as buttons, checkboxes, radio buttons, scrollbars, textboxes, etc., but you can also add menus, and their items. Adding the components to your application window is as simple as selecting the widget from the toolbar and then drawing it. Qt Designer does not stop there in terms of creating your interface. It also gives you the ability to define your signal/slot connections. This procedure is started by selecting the object that will emit the signal, then entering the name of the slot that the signal connects to.

When an interface has been created and the file is saved, Qt Designer will store your interface in a .ui file that is comprised of special XML code. It is a wise move on the part of Trolltech to use an open standard such as XML for their file format as the .ui files can then be repurposed for other uses such as XSL transformations to possibly link Qt Designer interfaces with web pages.

When the file is saved, a special tool called moc can be run on the file to convert it to the relevant C++ header and implementation file. Once the source code has been generated, you will have a boilerplate class for the interface available that can be used by re-implementing the class in your main code. You need to use polymorphism to use the class due to the fact that any modifications made to the generated class will be lost when you next generate the class. Using this method of re-implementing the class, you can then use the generated class and define your own slots of the same function name. This is a clever technique, and while a little confusing at first, gives the developer ultimate flexibility.

Qt Linguist

Qt Linguist is a tool for creating translations within Qt applications. The idea behind the tool is that you separate those who code the application from those who create translations. The traditional method of supporting multiple languages when developing software has been to implement multiple translations either within the code, or via a text file for each language. Qt prefers the more elegant technique of creating so called *translation files* that are created by the translators with Qt Linguist. Those files are then made use of by the developers in the Qt application. I like this technique because not only is it simple, but this kind of simplicity means that Qt developers can not only create multi-platform applications, but multi-platform, multi-language applications. I am sure that non-technical translators will appreciate this simplified method of dealing with translations, as well.

QMake

The final tool in the Qt toolbox (I will cover Qt Assistant in the next section) is QMake. This simple little tool lets you handle the building your applications easily. Many developers spread their code out over multiple source files, and traditionally developers have needed to edit Makefiles, and other build scripts, to get their applications to build. When you roll in the multi-platform nature of Qt with different compilers and build environments, this could get real challenging real fast. QMake seeks to simplify the process, and when run will generate a make file for building your application. I have some experience with the GNU tools to do this, such as automake, and QMake is a welcome change. I found QMake not only makes the build system seamless, but due to the fact that it is bundled with Qt, makes it even easier.

Documentation and Qt Assistant

One of the major strengths of Qt is its incredible documentation. As a developer, documentation is always something that is important to have available due to the fact that every intimate detail of the API may need to be used, and as such should be well documented. Luckily, the Qt documentation team have done an incredible job at not only creating an impressive reference manual for every minute detail of the API, but have also included a number of other features in the documentation. These include:

Qt Community
Information about mailing lists, newsletters, bug reporting and more.

Getting started
Details on how to begin programming with Qt. This section also includes two full tutorials, and many examples.

API Reference
A full and complete reference of every class, function, and definition in the Qt toolkit. The reference is concise, and easy to read.

Modules
Documentation, tutorials, and examples of each of the additional modules within Qt.

Overviews
This section provides a number of walkthrough, and discussion documents on various parts of the Qt toolkit, such as the Qt object modal, and signals and slots.

Porting and platforms
This section gives information, and details on supporting each of the different platforms for your project, compilation details, and specific platform notes.

Tools
This section provides documentation for each of the tools included with Qt, such as Qt Designer, Qt Linguist, Qt Assistant, and QMake.

Licenses and credits
This section provides information about the different licenses available with Qt.

The documentation available with Qt is in HTML format, and can be viewed in any web browser (as well as being available at http://doc.trolltech.com). Although a web browser suffices for basic viewing of the documentation, Qt includes a special tool for dealing with the documentation called Qt Assistant. Qt Assistant provides an interface for searching, and querying the documentation, and viewing it. Although a simple front-end to the documentation, Qt Assistant provides a more integrated method of reading the class reference/information instead of a web browser. I am impressed that Trolltech takes

BMS **THE LAW OFFICE OF BRADLEY M. SNIDERMAN**

Need help safeguarding your software?

If you're developing software, you need your valuable work protected with trademark and copyright registration, as well as Non Disclosure Agreements.

Then, when you are ready to sell it, you can protect yourself further with a licensing agreement.

I am an attorney practicing in Intellectual Property, Business Formations, Corporate, Commercial and Contract law.

Please give me a call or an e-mail. Reasonable fees.

23679 Calabasas Rd. #558 • Calabasas, CA 91302
PHONE 818-222-0365 FAX 818-591-1038 EMAIL brad@sniderman.com

documentation this seriously, and have committed to not only providing good documentation, but a tool to access it.

In use

Qt is a powerful toolkit, and has been carefully developed. When using the toolkit, you always get a true feeling of quality with what you are doing. This is largely from the tried and tested development model that has been created by Trolltech. Using C++ for graphical development is a natural choice due to its OOP philosophy, and it is very rare that you are in a position where you can see no elegant way of doing something. The available number of classes in Qt is impressive, and there is a convenience class for most common processes involved with modern software development. Not only do the classes allow you to create functionality easily, but each class has a rich API, consisting of a number of methods that are useful in most situations.

Qt Designer is an integral part of the Qt system, and generally works well for most applications. A few releases back Qt Designer was considerably more primitive than it is these days, and the latest version of Qt Designer is quite a mature and useful tool. While not a fully RAD tool (in the sense that it does not actively generate code that is embedded directly in your project), it is about as RAD as you want it to be – it generates the code, and *you* put it into your project. This prevents the kinds of spaghetti code problems that are often associated with RAD tools.

Although it is possible to review and evaluate Qt in a sterile environment where only Qt is used, in the real world Qt faces stiff competition from the likes of Java, Cocoa, GTK, and others. From my experience, Qt stands up well to these competitors, and offers a compelling, and in many cases, superior product. The main areas that many developers seem to look at when evaluating a product is its ease of use, functionality, and stability. From my testing, Qt seems to stand up on all of these points and I cannot really think of many areas in which Qt would be unsuitable for development. One testament to the nature of Qt, in my opinion, is the extent to which it is used. Because of the dual licensing of the toolkit, and the availability of a GPL version, Qt has been used by the UNIX based KDE project (www.kde.org). The sheer scope of that project is a good demonstration of the ability of Qt as a toolkit. KDE can be installed on Mac OS X using the Fink system (fink.sourceforge.net).

New in 3.2.2

Qt 3.2 has implemented a number of new features, in addition to the already impressive set of the 3.1 release. These new features include:

New Splashscreen class
This new class provides a contemporary splashscreen class that can be used to show a splash screen while your program is loading. The class will also allow you to display status messages in the splash screen.
New toolbox widget
A new widget has been added to create a toolbox that has collapsible areas. The widget was originally used in Qt Designer, and not available as a common class. Now it is, and can be added using the Qt Designer interface.
Improved menu editor in Qt Designer
The menu editor in Qt Designer has always worked fine, but not worked as intuitively as you might have expected. It has been revamped in 3.2.2.
Thread local storage
Multi-threaded applications can be written in Qt, and a new class has been added to store thread variables between threads.
Input masks
Input masks for controlling what input is added to a text entry box has been added. This makes it easier to validate user input.
Improved SQL support
With the addition of a DB2 driver, there has been a revamp of the SQL support in Qt, and data aware widgets work better with queries now.
Improved indic and syriac support
Support for right to left languages has been improved, with full support for indic and syriac languages.
Improved printer setup dialog
Additional functionality has been added to the printer setup dialog.

There was also a long list of user and developer submitted bugs that were fixed with this release, and various changes to particular platform versions of Qt.

Conclusion

Qt is a great toolkit. I am impressed with the technical structure of the software, the feature set, and the documentation that is included. Qt seems to offer a nice combination of hard core power, and RAD development. C++ is a good choice for a language to base the toolkit on, although it would be nice to see bindings for Java, or possibly even PHP. The toolkit not only provides a good range of graphical components, but also provides high performance data handling classes, and nice additions, such as data aware database widgets and classes, OpenGL support, and other features. The addition of Qt Assistant, Qt Linguist, and particularly Qt Designer, are sensible choices by Trolltech. Qt Designer, in particular, rapidly increases development speed.

If you are looking for a toolkit where you can write your code once, and run it on a number of different platforms, Qt is well worth looking in to. In these days of Windows, Linux, and Mac OS X, using a toolkit such as Qt makes sense. It makes particular sense for those developers creating free clients, and software to give away, as the GPL edition provides equivalent functionality to the commercial version, with only a change in licensing. The commercial version licensing allows you to distribute closed source applications commercially.

I look forward to seeing how Trolltech will expand Qt in the future, and look forward to seeing more Qt applications running on all my computers.

COVER STORY

By John A. Vink

Performance Sampling

Making code faster through introspection

Do It

Profiling your code is essential. You can't speed up your code if you don't know what is taking so long. You might think you know where the slowdown is, but most likely you'd be surprised. Engineers from the Safari team recount that they had a perfect record of incorrectly predicting what was slowing down their code. Only after doing some profiling, they discovered the real bottlenecks.

The process of profiling is:

1. profile your code
2. find the parts of the profile that belong to you and take significant amounts of time
3. optimize
4. lather, rinse, repeat

Here I am going to discuss the first two steps of profiling. You should already be familiar with "repeat". The second and following times through this loop you also need to see if the changes you made really did make things faster.

What are you talking about?

Sampling can be done from a command line tool or from a GUI application.

First, let's talk about what sampling actually does.

Sampling is finding out what your application is doing at any given time. About every 10 ms your application is asked, "What are you doing now? How about now? And now?" Your application responds by giving a stack trace each time. These are called samples. When the sampling period has completed, the results are summarized into a call graph.

Actually, that's just a simple way to conceptualize it. What's really happening is that the sampling application suspends the sampled application at periodic times. While the sampled app is suspended, the sampling app walks the stack for each of the sampled process' threads to ascertain the stack trace.

As an aside, sampling is very useful when your application appears hung. You can sample your hung application to see exactly where it is hung, giving you clues about how to fix it.

So let's imagine you sampled for 5 seconds, which would mean 500 samples when sampling every 10 ms. When sampling the main thread, the main() function is going to appear near the top of the stack since it's part of that thread's entry point. So it'll show up 500 times. Let's say you only have 2 functions in main - KindaQuick() and KindaLong(). KindaQuick() might show up 100 times, and KindaLong() 400 times. So your sample log will show main at 500 samples, and inside that, it will show KindaLong() at 400 samples and KindaQuick() at 100 samples. It would look something like this:

```
500 main
  400 KindaLong
  100 KindaQuick
```

Some things to note about samples is that if you have a function that can complete between two samples and you call it just once, then it might not show up in your sample log. Because it started after sample n, and completed before sample n + 1, no samples will show this function. But if you call that function a bunch of times, then chances are it will show up in your sample. This shouldn't be of much concern since if your function runs so quickly to be invisible to samples, there probably isn't much opportunity for optimization.

If your thread is sleeping, it is still being sampled. Sampling doesn't concern itself with actual CPU time used. It will look like a function is being really inefficient because it

John A. Vink is one of Apple's most gifted engineers. He currently does performance analysis on code that you, the user, run constantly every day. He hopes you'll read this and make his job easier. It's possible to email him at vink@apple.com.

shows up in so many samples, but that's because the thread is just sitting around waiting for a reason to wake up. Sleeping threads are a good thing since they don't take up any CPU time. Your application would be really efficient if all its threads were always sleeping, although your application wouldn't do much.

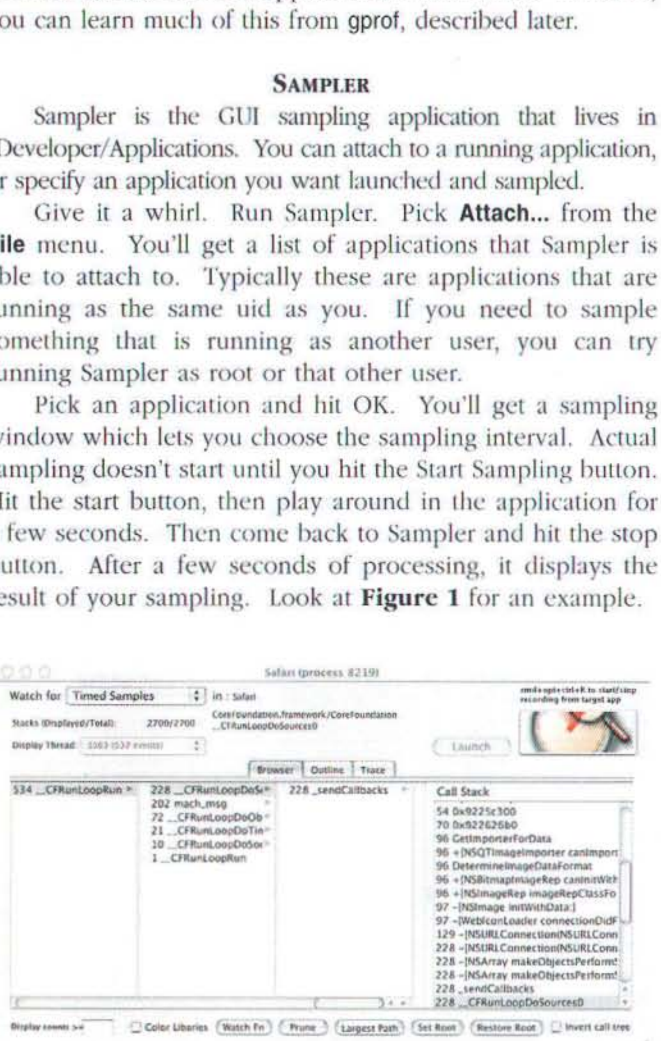
Sampling doesn't tell you when a function appeared in a stack trace - only how often. The only "when" information you can learn is which function called the function you're interested in at a particular point in the sample. You also can't tell how many times a function was called, only the number of times that the function appeared in a stack trace. However, you can learn much of this from gprof, described later.

SAMPLER

Sampler is the GUI sampling application that lives in /Developer/Applications. You can attach to a running application, or specify an application you want launched and sampled.

Give it a whirl. Run Sampler. Pick **Attach...** from the **File** menu. You'll get a list of applications that Sampler is able to attach to. Typically these are applications that are running as the same uid as you. If you need to sample something that is running as another user, you can try running Sampler as root or that other user.

Pick an application and hit OK. You'll get a sampling window which lets you choose the sampling interval. Actual sampling doesn't start until you hit the Start Sampling button. Hit the start button, then play around in the application for a few seconds. Then come back to Sampler and hit the stop button. After a few seconds of processing, it displays the result of your sampling. Look at **Figure 1** for an example.



```
534 __CFRunLoopRun      228 __CFRunLoopDoS...   228 _sendCallbacks
                        202 mach_msg
                        72 __CFRunLoopDoOb...
                        21 __CFRunLoopDoTim...
                        10 __CFRunLoopDoSo...
                        1 __CFRunLoopRun
```

```
Call Stack
54 0x9225c300
70 0x922626b0
96 GetImporterForData
96 +[NSQTImageImporter canImport
96 DetermineImageDataFormat
96 +[NSBitmapImageRep canInitWit
96 +[NSImageRep imageRepClassFo
97 -[NSImage initWithData:]
97 -[WebIconLoader connectionDidF
129 -[NSURLConnection(NSURLConn
228 -[NSURLConnection(NSURLConn
228 -[NSArray makeObjectsPerform
228 -[NSArray makeObjectsPerform
228 _sendCallbacks
228 __CFRunLoopDoSources0
```

Figure 1. *Main Sampler window.*

As you click on function names in the left column view, the next column to the right will populate showing all the functions called by the function you just clicked along with the number of samples for each. The right scroller will show you the stack trace up to that function, and the highest sampling functions after that. You can see in this figure that we've drilled down to __CFRunLoopDoSources(). You can see exactly where its parent, __CFRunLoopRun, spent all of its 534 samples. 202 samples were in mach_msg, which, if that path were followed, would reveal that the thread was sleeping. All of the time spent in __CFRunLoopDoSources() was spent in _sendCallbacks. The remaining 104 samples from __CFRunLoopRun were shared among __CFRunLoopDoObservers, __CFRunLoopDoTimers, __CFRunLoopDoSource1, and __CFRunLoopRun.

If you were tracking performance problems, you want to investigate the functions that are taking the most time, ignoring the samples that are sleeping. Keep drilling down until you see something that surprises you. 534 samples in __CFRunLoopRun is not surprising, and neither is 228 samples in __CFRunLoopDoSources, but perhaps 97 samples in WebIconLoader might be, so if that's the case, that's what you want to check out.

SAMPLE

sample is the command line tool that allows you to sample a process. This can be useful if you're remotely connected to the machine.

To sample a process, you invoke sample with the PID of the process you're interested in, and the number of seconds to sample for. You can optionally provide the duration between samples. So, first get the PID of the process you're interested in:

```
[vinkjo:~] jav% ps -aux | grep MyApp
jav     452   0.0  2.2    99616  22624  ??  S     Sun03PM
2:55.17 MyApp
jav   1696   0.0  0.0     1416    308 std  S+     5:49PM
0:00.00 grep MyApp
```

So now you know the PID you are interested in is 452. Now run the sample command:

```
[vinkjo:~] jav% sample 452 5
Sampling process 452 each 10 msecs 500 times
Sample analysis of process 452 written to file
/tmp/MyApp_452.sample.txt
```

Opening the resulting sample file will reveal that it looks something like this:

```
Analysis of sampling pid 452 every 10 milliseconds
Call graph:
    500 main
      400 KindaLong
        400  BlockMoveData [STACK TOP]
      100 KindaQuick
        100  memcpy [STACK TOP]

Sort by top of stack, same collapsed (when >= 5):
        BlockMoveData [STACK TOP]          400
        memcpy [STACK TOP]          100
```

In this hypothetical example we see that KindaLong took 4 times longer than KindaQuick. Perhaps this surprises us since both functions copy the same amount of data. If that's true, we can see that memcpy is much faster than BlockMoveData for the type and size of data we're giving it.

The sample shows **[STACK TOP]** to show when a sample shows that particular function at the top of the stack. This means, at the time the sample was taken, the code in that function was executing - not code in any other function that might be called from it.

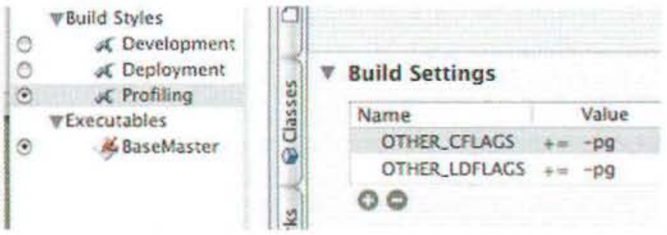
You can open the result of the sample command line tool in the Sampler 2.0 GUI application. You can select the sample file from the **Open...** dialog in Sampler, or open it from the command line like this:

```
[vinkjo:~] jav% open -a Sampler /tmp/MyApp_452.sample.txt
```

GPROF

Sampler and sample watch your code while it's running. For gprof, you run your code with profiling compiled and linked in, and when you're done, you use gprof to analyze the results. This allows you to profile command line tools and quickly running applications.

Using gprof requires you to rebuild your code. Because you need to have your code recompiled to take advantage of gprof, it might not be suitable when you're using a lot of third party frameworks whose code you can't recompile. Make a new build style and set the OTHER_CFLAGS and OTHER_LDFLAGS as shown in **Figure 2**.



Name	Value
OTHER_CFLAGS	+= -pg
OTHER_LDFLAGS	+= -pg

Figure 2. *Setting compiler options in Project Builder*

When your program completes, a file named gmon.out will be created in the current working folder from where you launched the application. This can be confusing, since if you launched it from the Finder, the gmon.out file will appear at /.

After you get your gmon.out file, you need to process it with gprof into something readable. To do that, run gprof something like this:

```
> gprof /BuildResults/MyApp.app/Contents/MacOS/MyApp
gmon.out > gprof.out
```

This will give you a report in the file gprof.out. There are two main sections to this report - the Call Graph and the Flat Profile.

The Flat Profile looks something like this:

```
granularity: each sample hit covers 4 byte(s) for 1.56% of
0.64 seconds

  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 12.5      0.08      0.08
_objc_msgSend [1]
  4.7      0.11      0.03
_DoLigatureXSubtable [2]
  3.1      0.13      0.02
_CFHash [3]
  3.1      0.15      0.02
__class_lookupMethodAndLoadCache [4]
  3.1      0.17      0.02
_objc_getNilObjectMsgHandler [5]
  3.1      0.19      0.02
_pthread_getspecific [6]
  1.6      0.20      0.01
+[NSDictionary dictionaryWithObjectsAndKeys:] [7]
  1.6      0.21      0.01                                   -
[NSLayoutManager defaultLineHeightForFont:] [8]
  1.6      0.24      0.01                                   -
[NSString isEqual:] [11]
  1.6      0.25      0.01                                   -
[NSUnarchiver decodeValuesOfObjCTypes:] [12]
  1.6      0.27      0.01
_CFAllocatorDeallocate [14]
  1.6      0.28      0.01
_CFDictionaryGetValue [15]
  1.6      0.29      0.01
_CFRelease [16]
  1.6      0.30      0.01
_CFRetain [17]
.
.
.
  0.0      0.64      0.00       20     0.00      0.00
__ZN13BaseConverter15GenericSetValueEtPc [18043]
  0.0      0.64      0.00       10     0.00      0.00  -
[ConverterView textFieldType:] [52]
  0.0      0.64      0.00        5     0.00      0.00  -
[ConverterView textDidChange:] [53]
  0.0      0.64      0.00        5     0.00      0.00  -
[ConverterView updateFieldsWithNewNumbers:] [54]
```

This shows the amount of time spent in each function, sorted in decreasing order by the number of seconds actually spent in each function (as opposed to time spent in it and the functions that it calls). Then it is sorted by the number of calls (this is only available for sources compiled with the -pg flag. So, your sources, not the frameworks), and then alphabetically by name.

The % time is the percentage of total execution time that your program spent in this function. The cumulative seconds is the amount of time that was spent running this function plus any function that it calls. If the number of calls for a function are available, you can discover the number of milliseconds spent in just this function per call (self ms/call), and the number of milliseconds spent in this function plus any functions it calls per call (total ms/call).

Here I can see that the C++ function BaseConverter::GenericSetValue() gets called 20 times. If this is more than I expect, then I should look into why it's being called so many times. You can see that the flat profile can tell you how many times a particular function was called, which is not easy to do with the output from sample, and you can also see the amount of time spent in an individual function compared to how long was spent in the functions that that function called.

It's important to note when a function appears to take a long time to execute because the function itself is slow or because it is called a large number of times. In the above example, _objc_msgSend comes out as the biggest "time sink", which may lead you to believe that it is the performance issue. When in fact, it probably isn't. The performance issue, if any, is likely to be that some code gets executed too much that happens to call _objc_msgSend a lot, and instead of focussing on speeding up the leaf routine, one should find out why the leaf routine is called so much. In your sources that you compile with the -pg flag, this will be more obvious since you get the call count, but keep this in mind for functions that you don't get the call count.

The other part of the gprof report is the Call Graph, which looks something like this:

```
granularity: each sample hit covers 4 byte(s) for 1.56% of
0.64 seconds

                                          called/total
parents
index  %time     self descendents  called+self     name
index
                                          called/total
children

                     0.00         0.00        5/10            -
[ConverterView textDidChange:] [53]
                     0.00         0.00        5/10            -
[ConverterView updateFieldsWithNewNumbers:] [54]
[52]      0.0    0.00         0.00       10           -
[ConverterView textFieldType:] [52]

-----------------------------------------------

                     0.00         0.00        5/5
__nsNotificationCenterCallBack [85241]
[53]      0.0    0.00         0.00       5
[ConverterView textDidChange:] [53]
                     0.00         0.00        5/10            -
[ConverterView textFieldType:] [52]
                     0.00         0.00        5/5
__ZN13BaseConverter14SetUnsignedDecEm [18045]
                     0.00         0.00        5/5             -
[ConverterView updateFieldsWithNewNumbers:] [54]

-----------------------------------------------

                     0.00         0.00        1/1
__start [85480]
[18052    0.0    0.00         0.00       1              _main
[18052]

-----------------------------------------------
```

Using the call graph, you can see which functions call a particular function, and also see what functions a particular function calls. Looking at the first entry, we can see that -[ConverterView textFieldType:] is called a total of 10 times - 5 times from -[ConverterView textDidChange:] and 5 times from -[ConverterView updateFieldsWithNewNumbers:]. Either -[ConverterView textFieldType:] did not call any other functions, or the functions that it did call were not compiled and linked with the -pg flag.

In the next entry, we can see the functions that -[ConverterView textDidChange:] called. It called -[ConverterView textFieldType:] 5 times out of the 10 times that the function was called throughout the program execution. It also called BaseConverter::SetUnsignedDec and -[ConverterView updateFieldsWithNewNumbers:] each 5 times.

With the results you get from gprof, here are some of the things you should be looking for:

1. Look for functions that use up a lot of self ms/call in the flat profile. A lot of time is spent in these functions, and the amount of time can not be blamed on other functions that it calls.

2. Take a look at the number of calls that your functions get. If they are larger than you expect, track down why they are larger than you expect. Some functions may be called redundantly.

3. Scan over the numbers and see if anything looks surprising or slightly unexpected. A big part of optimization entails looking for things that do not look right.

Which Functions to Optimize

Here are some ideas for finding which functions you should spend some attention on:

1. If a function takes a long time to execute but only executes once, then tuning that function's code is the best thing you can do. If a function gets run millions of times but spends little time executing, then the best thing you can do is get rid of the need to call it millions of times.
2. Scan your results to find "things that make you go hmmmm..." Surprising results means things aren't operating the way you had anticiapted. This could mean some design issues with your algorithm, some functions are more expensive than you had anticipated, or just implementation mishaps.
3. Go for the biggest bang. You may have a terribly inefficient function, but if it only takes up 0.1% of the time, then the biggest gain you can possibly get is 0.1%. Go after the function that takes 10% instead.

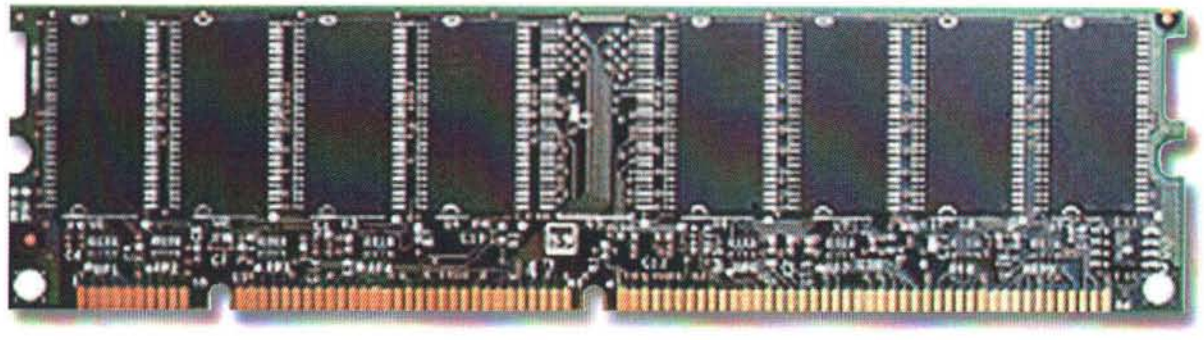
Summary

Don't postulate at what's wrong. Look at what's wrong.

References

For additional information, see Inside Mac OS X : Performance. More information on gprof is available at <http://www.gnu.org/manual/gprof-2.9.1>. Thanks to Yan Arrouye, Robert Bowdidge, Scott Boyd, and John Wendt for reviewing this article.

There are some things in life you can never have enough of...



Built-to-order. Lifetime Guarantee. Competitive Pricing.

DEV DEPOT®

www.devdepot.com/memory

Voice: 877-DEPOT-NOW (877-337-6866) • Outside US/Canada: 805/494-9797
E-mail: orders@devdepot.com

By the MacTech Magazine Editorial Staff

"Yoyo" Power Supply Repair

How one goes about writing for the magazine

Problem

Apple's award-winning industrial design is world-renowned. One of the most eye-catching devices introduced with the original iBook was the so-called "Yo-yo" power adapter. Sleek and compact, it unfortunately came with a flaw; the vulnerability of the cable near where it connects to the computer. After repeated flexing (from 6 months to over a year) the braided ground portion becomes frayed and detaches inside the clear insulation. This leads to intermittent connections, then complete failure. We have experienced four of these failures, on my wife's iBook/300 and my Pismo/400. The first one was replaced under AppleCare, but the rest would require replacement at $65 each. (Used ones on eBay might be found for $35-$50).

After several months of being out-of-work (common these days for a high-tech software professional in Silicon Valley) and sharing our one functional unit between the two laptops, I finally took a look at the two broken ones we had left and decided to do a little surgery. What could go wrong? The result, after about 1.5-2 hours of careful work, is a functioning power adaptor. It's not pretty, but it should function for a while. I'd like to share the repair process.

Required Tools

Soldering iron and solder (electronic/rosin core); several pliers (slip-joint and needle-nose); an X-acto hobby knife with pointed blade; RTV silicon sealant; electrical tape and shrink-fit tubing; very steady hands and a keen eye doing close-up work. (I once worked as a professional jeweler, and have had much experience doing electrical work. YMMV..). One additional tool is a "third-hand," (http://store.yahoo.com/internettrains/exhotoexha.html) a hobby/craft device with clips to hold your work while you solder.

Disassembly

The first stage is disassembling your adaptor. You need to first cut away the clear covering on the connector; carefully cut lengthwise along the top side (away from the nub that fastens the cable end to the housing). Then cut around the connector so you can separate this piece into two sections; one will slip off the end of the connector, the other will slide back along the cable away from the end. Put these pieces aside so that they may be reassembled later. (**Figure 1**)

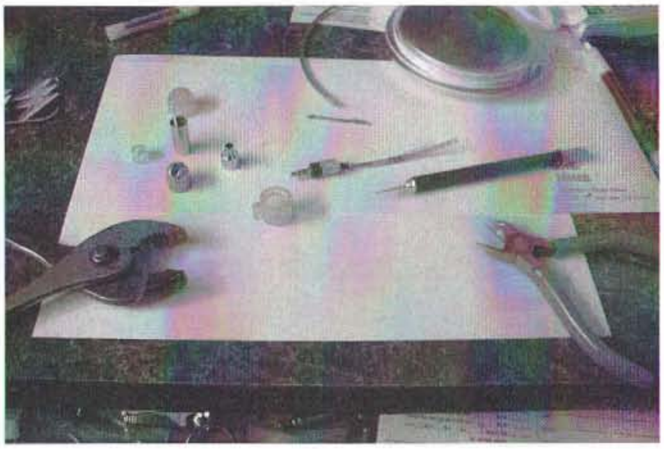


Figure 1. *Initial stage of disassembly*

(I apologize for not showing earlier stages of the disassembly; the idea to write the article occurred at this stage of the project).

Connector Surgery

Once you have the covering off, you will find a metal slug through which the cable threads. Pull this back to expose the tail end of the connector shell itself. Before disassembling the shell, you will need to de-solder the grounding resistor (**Figure 2**).

The author of *this article* has been working with computers for far longer than he would like to admit, and on Macs since late 1984. His wife, two grown children, both parents, and both (older & younger) sisters are all Mac owners as well. When NOT working or playing with their Macs, Mark and his wife enjoy the company of three Yorkshire Terriers, three cats, and five cockatiels. You can reach him at n_mark_jaffe@yahoo.com.

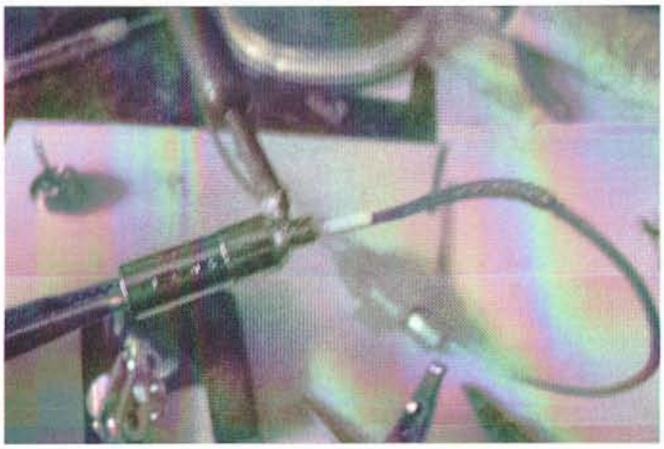


Figure 2. *De-solder resistor from connector end before pulling apart*

Now that you have the resistor de-soldered, bend the lead back carefully so that the connector end may be detached from the connector itself. A slight tap may be needed to separate the parts. Now the area where most of the work happens is exposed. (**Figure 3**).

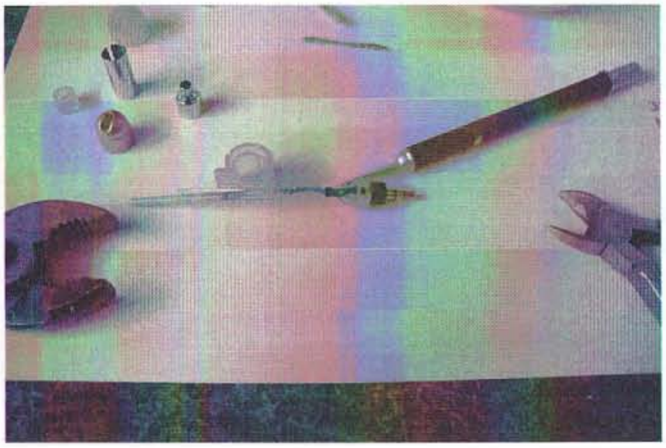


Figure 3. Connector innards exposed.

Here you will need to cut away all the clear "potting" material so that the wires may be de-soldered from the connector. Carefully cutting into the material with your X-acto knife, you may chip the material free from the connector while taking care to NOT slice into the resistor. This one piece needs to be intact when re-assembling your power-supply cable.

After you have cleaned out all the potting material (**Figure 4**), you will be able to de-solder the wires from the old (broken) end of the cable.

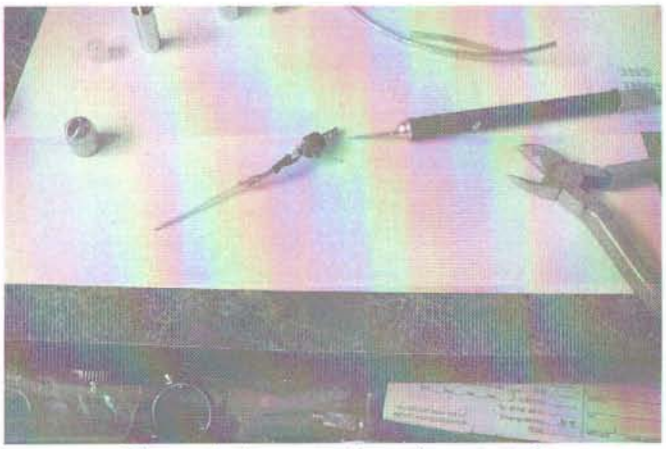


Figure 4. *Connector innards un-potted.*

Here is where the "third-hand" becomes a valued tool; you need to hold the soldering iron in one hand, have a tool in another hand to pull the wire free (tweezers or a metal pointed tool) and be able to hold the work being de-soldered so that you don't get burned.

REASSEMBLY

Now that you have removed the old wire from the connector, we can deal with the broken end of the cable to be reused. Before going further, you will have to thread back onto the cable the external parts of the assembly that were initially cut free. (You did cut them free carefully so they could be reused, and did not throw them out, right?) Make sure they go back in the order shown in **Figure 5**. (Additional note: one piece in the illustration did not belong; I had to remove it later. It is the piece with the "nub" that holds the cable end to the housing you wind the cord into). At this time you should also thread onto the cable a 1.5-inch segment of shrink-tubing which will later be used to protect the braided ground shielding.

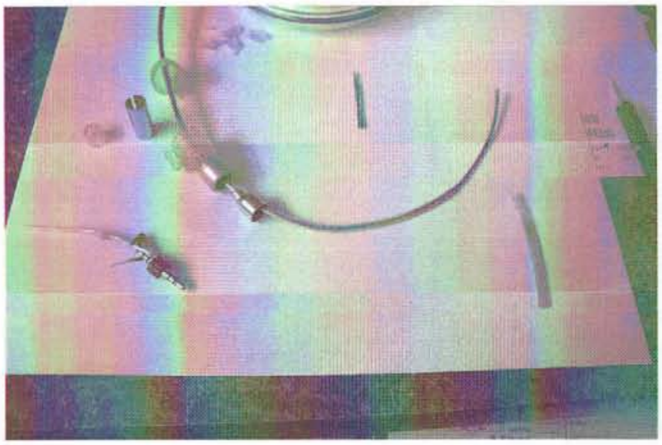


Figure 5. *Re-assembling the shell begins...*

Now that you have threaded those parts onto the cable, you need to prepare the end for re-soldering. Cut the end clean where the cable had gotten frayed, then strip back about two inches of the outside insulation. Carefully push about an inch of the braided ground layer back from the end, and twist it off to the side (**Figure 6**). Twist up about 1/2 inch of the braiding and lightly "tin" the end with solder (this will help to attach it to the connector). Twist up 1/4 inch of the inner conductor and "tin" the end.

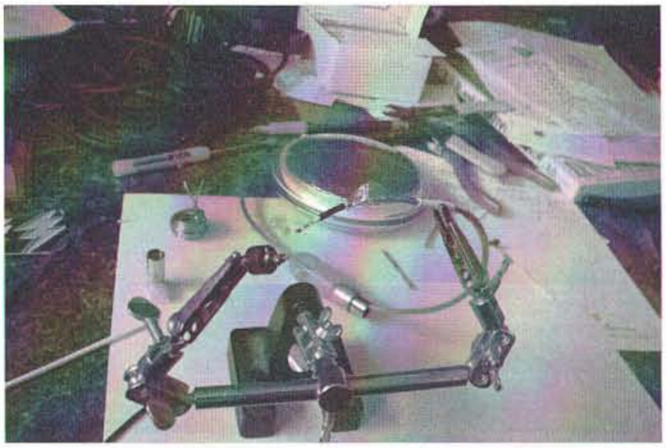


Figure 6. *Preparing the end for soldering.*

Now you should slip a 1/2-inch section of shrink-tubing over the inner conductor; then you are ready to attach the end to the forward-section of the connector, where it was attached before (**Figure 7**).

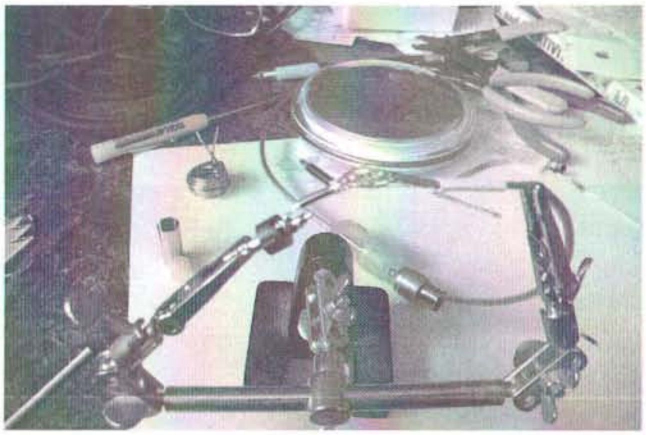


Figure 7. *Reattach center conductor.*

The tricky bit comes next. You will need to solder both the braided ground and one end of the resistor to the middle segment. (My apologies for not including a photo of this step).

Now that the ends are connected, you may begin to reassemble the shell. Slip the long tube over the end of the connector innards; there is one "right" way it goes on. You will see evenly spaced rectangular indentations on the end where the tip protrudes. And the tip should protrude about 1/8 inch beyond the end of the tube (**Figure 8**).

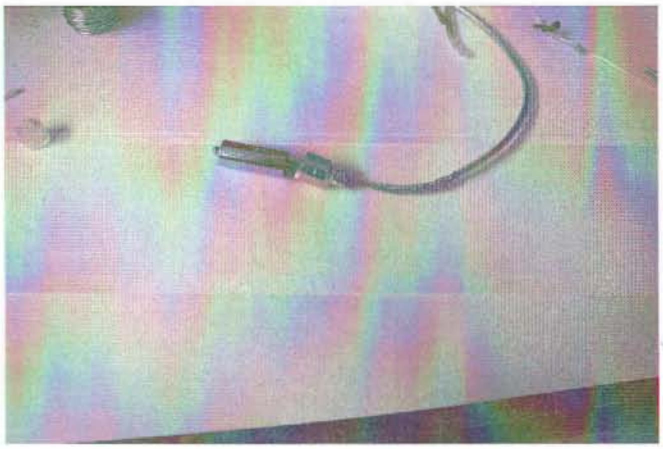


Figure 8. *Assemble connector*

Before you fit the inner end of the inside portion over the end tube, you will be repotting the soldered area with silicon seal (**Figure 9**). You should also take care that the tail of the resistor is threaded out the end of the cap back toward the cable; you will need to solder it onto the outside of the shell after reassembly.

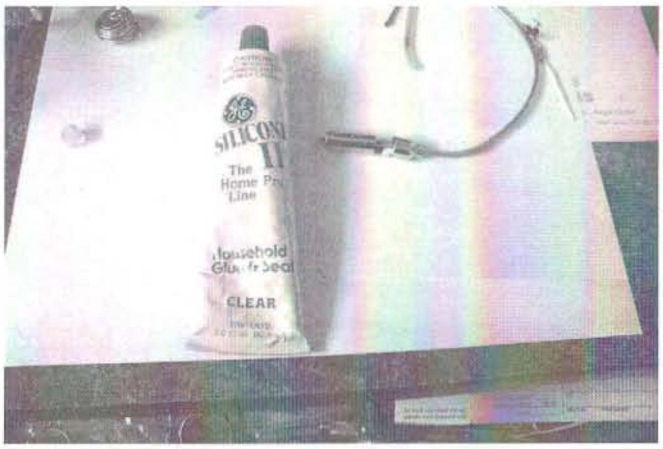


Figure 9. *Example of RTV (Room Temperature Vulcanizing) silicon seal*

Squeeze a small amount into the end of the tube and a little into the inner cap; then using a flat tool, pack the material into both ends of the connector. Slip the cap onto the tube and carefully, while holding the tube end with pliers, press-fit the cap onto the tube (**Figure 10**). You will see some of the sealant ooze out; this can be cleaned off later.

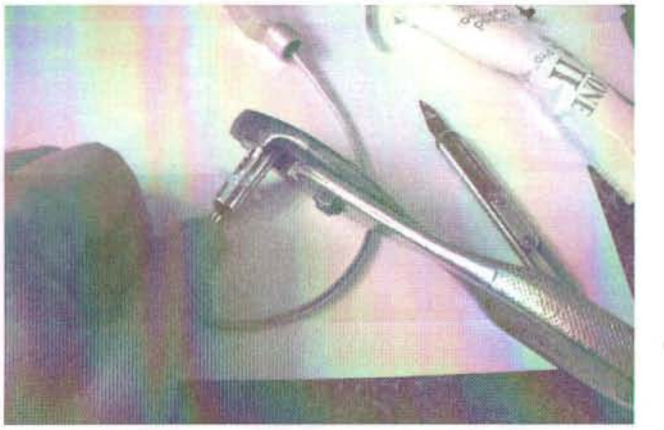


Figure 10. *Press-fit cap onto tube after packing with silicon seal.*

Let this assembly cure for a while, and get a cup of coffee. After it has cured, carefully bend the tail of the resistor back over the metal tip and solder it to the shell. (Refer to **Figure 2**).

Now you can slip the gray metal piece onto the end of the connector, and then slip the outer plastic section over the metal slug. Now slide the portion of the cover with the nub over the end of the connector, as shown in **Figure 11.**

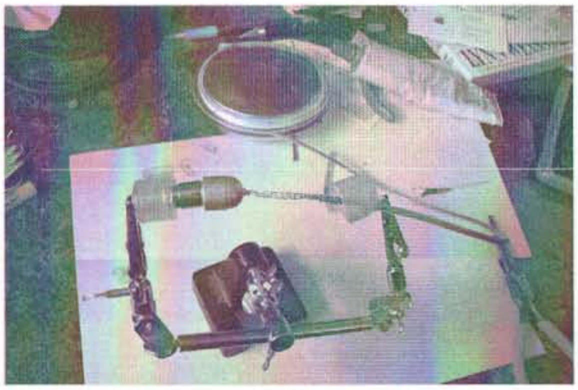


Figure 11. *Reassembly of outer cover.*

When the segments of the shell covering are pressed together, you can complete the repair by wrapping in electrical tape. Start at the very end and work toward the cable. Keep the tape tight as you go, as the tape will be a structural member of the assembly; it will serve to keep the shell together and provide shielding to the exposed section of braided ground.

TESTING

Now that you have completed reassembly, the moment of reckoning is at hand. Plug your power cord into the yoyo housing, and then into the wall socket. Now attach the repaired cable connector to your iBook or PowerBook. If your laptop is powered on and booted up, you should see the battery/power indicator showing it is charging/plugged in. Congratulations! You're an electronics repair technician!

Long Distance

3.9¢ Per Minute!

Straight 6 second billing increments

Excellent rates on intrastate, intralata/toll calls and international calling with no term contract.

Toll Free (800/888/877/866) service, same low per minute rate for incoming calls.

10 cents per minute calling card.

Detailed billing directly from Capsule Communications, a Covista Company.

Quality electronic and telephone customr support.
No monthly billing fee if you sign up for AUTOPAY billing option or if your bill is over $20.00 each month.

(NOTE: $1.00 billing fee is charged when your bill is under $20.00 for all non-Autopay customers.)

www.lowcostdialing.com

Andrew Troelsen, Minneapolis, Minnesota

A Primer on the C# Programming Language (Part One)

Exploring .NET development on Mac OS X

Topic Prelude

If you have read the previous installments of this series (except last month when I missed the deadline), you should be well aware that the .NET platform is language agnostic. While this is technically correct, I'd be lying if I did not admit that C# is currently the language of choice for a majority of .NET developers. Given this fact, my goal in the next two issues is to offer a tour of the major syntactical features provided by C#. Understand of course, that two articles cannot possibly examine each and every token and construct of a given language. In any case, once you have digested the material presented here, you should feel quite comfy with the core aspects of C# and be in a perfect position to follow along with the articles to come.

Recall the Pillars of OOP

All object languages must contend with the three 'pillars of object oriented programming' (OOP). First, an OOL (object oriented language) must address the concept of *encapsulation*. Simply put, encapsulation services provide a way for types to safely control access to their private data points. Next we have *inheritance*, which provides a manner in which exiting types may be reused and extended. Last but not least we have the third pillar of OOP termed *polymorphism*. This pillar of OOP allows related types to respond uniquely to the same method invocation. Given that C# is a modern OOL, you are correct to assume that C# provides complete support for the pillars of OOP, as well as a relatively new paradigm termed *aspect oriented programming* (AOP) which will be examined in a later article. Before we dig into the details of OOP using C#, let's check out basics of defining classes and allocating objects.

Defining Class Types in C#

Classes are the cornerstones of any OOL, in that they provide a template for the objects that comprise your application. In C#, class types are defined using (not surprisingly) the 'class' keyword. Like other languages in the C family, C# classes may support any number of overloaded constructors to allow an object to come to life in a particular state. To illustrate, assume we have a class named Car (defined in a file named car.cs), which defines three pieces of private data to represent a given automobile's pet name, current speed and make (**Listing 1**).

Listing 1.A simple C# class definition (car.cs).

```
using System;

namespace CSharpOOPExample
{
  // The Car class type.
  public class Car
  {
    // State data (could be defined as protected if desired).
    private string petName;
    private int currSpeed;
    private string make;

    // Constructor set.
    public Car(){}
    public Car(string pn, int cs, string m)
    {
      petName = pn; currSpeed = cs;
      make = m;
    }

    // Override the virtual method System.Object.ToString()
    // in order to display our state data.
    public override string ToString()
    {
      return string.Format(
        "Name: {0}; Speed: {1}; Make: {2}", petName,
        currSpeed, make);
    }
  }
}
```

The first point of interest has to do with the 'public' keyword used when defining the type. In C#, a non-nested type may be defined as public or 'internal' (the default). The distinction is made clear when you recall that types are contained within an assembly. Public types may be created and manipulated across

Andrew Troelsen is a seasoned .NET developer who has authored numerous books on the topic, including the award winning *C# and the .NET Platform*. In addition to authoring articles for MacTech, Andrew also writes a monthly column for *C# Pro* magazine where is examines the gory details of the C# programming language. He is employed as a full-time .NET trainer and consultant for Intertech Training (www.intertechtraining.com), but thinks he is still on the beaches of Mexico sipping funky drinks served in coconut shells. You can contact Andrew at atroelsen@mac.com.

assembly boundaries. On the other hand, internal types can only be created and used by the assembly that defines it. Thus, if you were to build a .NET assembly which contained three public types and two internal types, other assemblies would be completely unaware of the two internal types. Using this technique, assembly builders are free to define any number of internal helper types that are invisible to other binary units.

Next, you can see that we have defined a method named ToString() using the C# 'override' keyword. This brings up a very important point regarding the .NET platform: If you do not say otherwise, class types automatically derived form the cosmic base class, System.Object (introduced in the October 2003 installment of *Casting your .NET*). Like many base classes, Object defines a set of virtual methods that may be redefined (which is to say, *overridden*) by derived classes. Here, the virtual System.Object.ToString() method has been overridden in order to return a custom blurb of text which represents the current state of a given Car type (the default implementation of ToString() returns the type's fully qualified name). We'll examine the details of virtual members and member overriding a bit later in this article.

Finally, note that the Car class has two constructors. The role of our custom constructor is clear enough, in that it allows the object user to create a car in an initial state. The other constructor may seem a bit odd (depending on your exposure to other C based languages) in that it's implementation does nothing at all. C#, like Java and C++ always supply a freebee no-argument constructor (termed the default constructor) for each and every class definition. However, as soon as you define a custom constructor, the default, default constructor (pardon the redundancy) is removed. Therefore, if you with to allow objects of this type to be created with no arguments, you must redefine the no-argument constructor.

On a final constructor-related note, recall the C# automatically sets a type's state data to safe default values (quite *unlike* C++). The rules are quite simple:

Numerical data is set to 0 or 0.0.
Object references are set to null.
Booleans are set to false.

Given this language feature, we have no need to assign values to petName, currSpeed or make within the scope of the default constructor.

Allocating and (indirectly) Destroying Objects

We are now ready to create an application object (e.g., the class defining the Main() method) to allocate an auto or two onto the managed heap (**Listing 2**).

Listing 2. Creating Cars (carApp.cs).

```
using System;

namespace CSharpOOPExample
{
   // The App object.
  internal class CarApplication
  {
    // Program's entry point
    private static void Main()
    {
      // Make some Cars!
      Car noName = new Car();
      Console.WriteLine(noName.ToString());

      Car someCar = new Car("Zippy", 90, "BMW");

      // ToString() called automatically.
      Console.WriteLine(someCar);
    }
  }
}
```

In C#, the 'new' keyword is used to allocate an instance of a given class type onto a region of memory termed the *managed heap* (in fact, class types can only be heap allocated. Stack allocated data is achieved through the use of value types, as described in the next issue). In a similar fashion as the Java platform, .NET provides a garbage collector that is in charge of destroying unused objects when required. Given this, C# does not provide a corresponding keyword (such as 'delete') to explicitly destroy an object. However, C# does provide a syntax which looks suspiciously close to a C++ style destructor. Ponder the following updated Car type (**Listing 3**):

Listing 3. C# style 'destructors'.

```
// The updated Car class type
public class Car
{
  ...
  ~Car()
  {
    Console.WriteLine("This car is destroyed.");
  }
}
```

Like C++, C# destructors are syntactically denoted using a tilde prefix, however the similarities end here. First and foremost, a C# destructor is in reality a short hand notation for overriding the virtual System.Object.Finalize() method (to verify this for yourself, run your assembly through ildasm and check out how C# destructors are expressed in terms of CIL code). Next, remember that C++ memory management is quite deterministic in nature, in that we programmers are the individuals responsible for destroying heap allocated memory. In contrast, the .NET platform uses a non-deterministic finalization approach. In other words, you do not know exactly *when* the garbage collector will remove an object from the heap, only that the associated memory will eventually be reclaimed. When the object is indeed destroyed, the .NET runtime will ensure that your type's destructor is invoked if present.

As you might guess, there is much more that could be said about the .NET garbage collection process (such as programmatically controlling garbage collection using System.GC, implementing the IDisposable interface, object generations, etc), however the current explanation will suffice for now.

Encapsulation Support via Type Properties

Our current iteration of the Car type is dysfunctional to say the least, as we have not provided a way to get or set the state data after the time of construction! .NET programming languages prefer the use of type properties, rather than traditional getters and setters to honor the pillar of encapsulation. In the February 2004 issue you examined property syntax via JScript.NET, and will be happy to know that the act of defining properties in C# is quite similar. Listing 4 illustrates how to preserve encapsulation of the private petName member variable using a public property named Name (assume that the make and currSpeed member variables are encapsulated by additional properties; Make and Speed respectively).

Listing 4. C# property syntax.

```
public class Car
{
  private string petName;
...
  // C# Property syntax.
  public string Name
  {
    get { return petName; }
    set { petName = value; }
  }
}
```

Recall that the name of a property does not need to have any relationship to the name of the data point it is responsible for exposing. Also note the use of the 'value' token in the implementation of the property set scope. Truth be told, 'value' is not really a true-blue keyword in the C# programming language given that it is legal to define member variables or local variables named 'value'. However when this token appears in the context of a property setter, it is used to represent the incoming value assignment. Finally, it is worth noting that properties can be configured as read-only or write-only. Simply omit the get or set scope from the property definition.

For testing purposes, update your Main() method to change and obtain various data points (**Listing 5**).

Listing 5. Exercising our properties.

```
internal class CarApplication
{
 // Program's entry point
  private static void Main()
  {
    ...
    Car someCar = new Car("Zippy", 90, "BMW");
    someCar.Name = "Junior";
    Console.WriteLine("{0} is going {1} MPH.",
      someCar.Name, someCar.Speed);
  }
}
```

At this point you can compile your C# files into an executable assembly (don't forget to enable an SSCLI-aware Terminal). The commands listed in **Listing 6** will do the job nicely.

Listing 6. Compiling and executing our Car application.

```
DoSscli
csc *.cs
clix carApp.exe
```

Next up, we will examine how to build a set of related types using classic inheritance.

The Syntax of Inheritance

As mentioned, if you build a class type that does not explicitly specify a base class, your type will automatically derive from System.Object. However, when you wish to build class hierarchies you will no doubt be interested in deriving new types from existing class definitions. This is accomplished in C# using the colon operator. Let's create three child classes (SportsCar, MiniVan and JamesBondCar) that leverage our current Car type (assume each of these new types are within a file named childCars.cs and are wrapped within the CSharpOOPExample namespace). **Listing 7** defines the SportsCar type.

Listing 7. The SportsCar class type (childCars.cs).

```
// SportsCar IS-A Car.
public class SportsCar : Car
{
  public SportsCar(string pn, int cs, string m)
    : base(pn, cs, m) {}
  public SportsCar(){}

  public void TurboBoost()
  { Speed =+ 20; }
}
```

Beyond the fact that SportsCar is explicitly deriving from the Car type (and therefore inherits each of the public and protected members of it's base class), observe the use of the 'base' keyword in the custom constructor. As you can see, 'base' is dangling off the constructor definition by way of a single colon (which in this case is not marking the name of the base class). When the base keyword is used in this manner, you are specifying which constructor to call on the parent class when the derived type is created. Given that Car already has storage for the petName, currSpeed and make data points, we are explicitly calling the three-argument constructor of the parent. If we did not do so, the parent's default constructor would be called automatically, forcing us to set the private data points using the inherited public properties or possibly protected data (which is obviously less efficient).

As well, notice the implementation of the TurboBoost() method. Another bonus of property syntax is that they respond to the intrinsic operators of C#. Thus, rather than having to increase the SportsCar's speed using traditional accessor and mutator logic (**Listing 8**) we can use the more streamlined code "Speed += 20;".

Listing 8. Properties streamline traditional get / set logic.

```
// If we were not using properties...
public void TurboBoost()
{
  setSpeed(getSpeed() + 20);
}
```

Listing 9 details the MiniVan type, who's first point of interest is that the custom constructor passes three of the four incoming arguments to the base class, while assigning the fourth and final parameter to it's own custom point of data (numberOfKids). Also notice how MiniVan overrides the parent's implementation of ToString() to account for it's custom piece of state data. In this case, the 'base' keyword is *not* triggering a base class constructor using the 'dangling colon on the constructor' syntax, but simply calling a base class method within the scope of the method definition.

Listing 9. The MiniVan class type (childCars.cs).

```
// MiniVan IS-A Car.
public class MiniVan : Car
{
  public MiniVan(string pn, int cs,
    string m, int k)
    : base(pn, cs, m)
  {
    numberOfKids = k;
  }
  public MiniVan(){}

  private int numberOfKids;
  public int KidCount
  {
    get {return numberOfKids;}
    set { numberOfKids = value;}
  }

  public override string ToString()
  {
    return string.Format("{0}: kids: {1}",
      base.ToString(), numberOfKids);
  }
}
```

Finally, **Listing 10** illustrates the final automobile, JamesBondCar. Note that JamesBondCar extends SportsCar, which in turn extends Car (which extends System.Object).

Listing 10. The JamesBondCar class type (childCars.cs).

```
// JamesBondCar IS-A SportsCar
// which IS-A Car.
public class JamesBondCar : SportsCar
{
  public JamesBondCar(string pn, int cs, string m)
    : base(pn, cs, m) {}
  public JamesBondCar(){}

  public void DiveUnderWater()
  { Console.WriteLine("Diving under water!");}
  public void Fly()
  { Console.WriteLine("Taking off into the air!");}
}
```

Needless to say, our JamesBondCar is able to dive under water and fly into the air to escape the current enemy at hand. Further, given that this type does not add any new member variables to the mix, the parent's ToString() implementation will fit the bill nicely.

Building a Array of Car types

At this point the Main() method may be updated to exercise each of these new derived classes. To provide a more interesting example however, let's create an array of Car types. As you are most likely aware, most OOLs (including Java and C++) allow you to store a derived object in a base class reference (e.g., and implicit cast). This is legal given the 'IS-A' relationship enforced by classical inheritance. Given this fact, update Main() to create an array of Car-compatible types and iterate over the array using the C# 'foreach' keyword to invoke each object's ToString() implementation (**Listing 11**).

Listing 11. Creating an array of Car-compatible types.

```
// Make an array of Car types.
Car[] allTheCars = new Car[3]
 { new SportsCar("Zippy", 85, "Audi TT"),
   new MiniVan("KidMobile", 55, "Caravan", 10),
   new JamesBondCar("QMobile", 120, "*Classified*")};

// Print out the number of cars in array.
Console.WriteLine("You have {0} cars:",
  allTheCars.Length);

// Call each auto's ToString() method.
foreach(Car c in allTheCars)
 Console.WriteLine(c.ToString());
```

A few points of interest. In C#, arrays are declared using the C-like square bracket notation. Here we have created an array of Car types named allTheCars. The 'new' keyword used when declaring the array is *not* creating any particular Car type, but rather the underlying System.Array in the background. Given that C# arrays always derive from the System.Array base class, each array has access to each of the public members (such as the Length property seen in the previous code segment).

Once we have allocated a System.Array capable of holding Car-compatible types, we can leverage the curly-bracket shorthand notation to fill the array with sub-elements at the time of creation. If you would rather, you are free to allocate and initialize an array on an item-by-item basis (**Listing 12**).

Listing 12. Creating an array of Car-compatible types (the long way).

```
// Long hand array creation.
Car[] allTheCars = new Car[3];
allTheCars[0] = new SportsCar("Zippy", 85, "Audi TT");
allTheCars[1] =
  new MiniVan("KidMobile", 55, "Caravan", 10);
allTheCars[2] =
  new JamesBondCar("QMobile", 120, "*Classified*");
```

The Details of C#'s 'foreach' Keyword

Speaking of System.Array, if you were to look up the formal definition of System.Array using any of the tools that ship with the SSCLI (see Feb 2004) you will find that this class type implements an interface named System.Collections.IEnumerable. System.Collections.IEnumerable defines a single method named GetEnumerator(), which returns yet another interface named System.Collections.IEnumerator. This interface provides a way for a type to iterate over contained sub-items using three members:

- Current : This property returns the item current 'pointed to'.
- Reset() : This method resets the internal indexer's position to the first item.
- MoveNext() : As you would guess, this advances in the internal indexer by one. Returns false when the end of the list has been reached.

So, why do we care about IEnumerable and IEnumerator? Well, the 'foreach' keyword of C# is preprogrammed to obtain these underlying interfaces to traverse the sub-objects of the array being iterated over. Given that all of these sub-objects have a ToString() implementation, we can safely invoke each auto's custom version. Be aware that System.Array is not the only type which implements the necessary interfaces required by the foreach construct. Most of the class types found within System.Collection support similar infrastructure, and if you wish to build a strongly typed custom collection, you can implement these interfaces directly to traverse custom types using 'foreach'.

The ABCs of Polymorphism: Virtual and Abstract Members

The final pillar of OOP to examine is polymorphism, which you have already begun to leverage when you overrode the virtual System.Object.ToString() and System.Object.Finalize() methods in your custom class types. As mentioned, classical polymorphism is the trait of OOP that allows hierarchies of types to responding uniquely to the same message (a.k.a., method invocation). Polymorphism is supported in C# using four simple keywords:

- Base: As already seen, 'base' allows you to trigger a parent's method / constructor.
- Virtual: This keyword allows you to define a base class method that has a default implementation, but may be overridden by a child class if required.
- Abstract: This keyword allows you to define an abstract base class (ABC) as well as abstract methods. Recall that abstract classes cannot be directly created, but can be used to hold references to derived types. Also recall that abstract methods do *not* have a default implementation, and therefore derived types must provide a concrete implementation, or define themselves as abstract classes as well.
- Override: This keyword allows a derived class to redefine a virtual method as well as implement an abstract member.

To inject some polymorphic activity into our existing application, let's add an abstract method to our Car base class called PrintBumperSticker(). Be aware that when a class defines an abstract member, the class must also be marked as abstract (**Listing 13**).

Listing 13. The abstract Car type.

```
//The abstract Car class type
public abstract class Car
{
   // All derived classes must
   // implement this member or become
   // abstract types as well.
   public abstract void PrintBumperSticker();
...
}
```

Given that Car has now been redefined as an ABC, it is a compile time error to directly create Car types. However Car (and ABCs in general) still has a very useful purpose, in that it defines all of the common functionality for derived types. Car has been further updated with a single abstract member, and therefore each and every derived class in now required to provide an implementation of this member (if you do not, you are issued compile time errors).

To rectify this issue, update SportsCar, MiniVan and JamesBondCar as you see fit using the 'override' keyword. **Listing 14** shows one possible implementation for the MiniVan type.

Listing 14. MiniVan's PrintBumperSticker() implementation.

```
public class MiniVan : Car
{
  public override void PrintBumperSticker()
  {
    Console.WriteLine
      ("All my money and Kids go to the U of Mn.");
  }
...
}
```

Runtime Type Discovery using C#

Now that we have a polymorphic interface defined by our base class, we can update our Main() method to invoke each type's custom implementation (**Listing 15**).

Listing 15. Polymorphism at work.

```
foreach(Car c in allTheCars)
{
  Console.WriteLine(c.ToString());
  c.PrintBumperSticker();
  Console.WriteLine();
}
```

Understand that the code within the foreach block is only legal because of the defined polymorphic interface. We can rest assured that all types have an implementation of ToString() given the fact that all classes ultimately derive from System.Object, and that all descendents of car must implement the abstract PrintBumperSticker() method. However, what if we wish to call specific members of the JamesBondCar, MiniVan or SportsCar types? If you look closely at the foreach syntax, you can see that we are using a base class reference to represent each sub-object. Therefore, the following would be a compile time error (**Listing 16**).

Listing 16. Can't directly access derived type members from a base class reference!

```
foreach(Car c in allTheCars)
{
 ...
 // Nope! Car does not define a KidCount
 // property! Compiler error!
  int numberOfKids = c.KidCount;
}
```

When you need to dynamically discover if a given type is comparable with a given base class (or interface), C# provides two keywords. Ponder the following update (**Listing 17**).

Listing 17. Runtime type discovery in C#

```
foreach(Car c in allTheCars)
{
  Console.WriteLine(c.ToString());
  c.PrintBumperSticker();
// Is 'c' a JamesBondCar?
  if(c is JamesBondCar)
     ((JamesBondCar)c).DiveUnderWater();

// Is 'c' a MiniVan?
  MiniVan m = c as MiniVan;
  if(m != null)
     Console.WriteLine("I have {0} screaming kids!",
        m.KidCount);
  Console.WriteLine();
}
```

The 'is' keyword is quite helpful in that it returns a System.Boolean that denotes if the current object is compatible with a given base class (or interface type). If the test succeeds, you can make a safe explicit cast (using the familiar C-style casting syntax) to access the underlying functionality. The 'as' keyword is similar, however 'as' will return a null object reference if the types being tested are incompatible. Therefore, when performing a runtime check for type compatible using the 'as' keyword, be sure to test for null before casting!

On a final casting related note, you can make use of one additional construct to check for type compatibility: structured exception handing (**Listing 18**).

Listing 18. Manually handling an InvalidCastException.

```
// Is 'c' SportsCar compatible?
try{
  ((SportsCar)c).TurboBoost();
}
catch(InvalidCastException ex)
{
  Console.WriteLine("OOPS!  Not a SportsCar.");
  Console.WriteLine(ex.Message);
}
```

Here, we are making use of structured exception handling to attempt to cast current item pulled from the array of Car-compatible types into a SportsCar. If the cast fails, the runtime will throw a System.InvalidCastException type. Given that all exceptions derive from a common base class named System.Exception, we are free to make use of any of the inherited members to display information about the error in question (such as the Message property). See online help for complete details of System.Exception.

Interface-Based Polymorphism

Excellent! At this point you have a solid understanding of how C# contends with the mighty pillars of OOP and gained some insights into runtime type discovery, explicit casting and structured exception handling along the way. To complete this article, we will shift away from our examination of class types and examine the role of interface types (remember that .NET defines five possible types from the set {class, interface, structure, enumeration, delegate}).

Interface types, in a nutshell, are a named collection of abstract (and only abstract) members. On its own, an interface is of little use, given that you cannot create an instance of an interface variable. However, when an interface is implemented on a given class (or structure) you are able to bind a set of behaviors to the type in question. The power of interface-based programming becomes crystal clear when you understand that these types allow you to breath polymorphism into types found in *different class hierarchies*. Let's see a complete example.

Assume you have created some new class type modeling UFOs. Given that UFOs (presumably) are not automobiles, the UFO base type will derive directly from System.Object, whereas MotherShip extends UFO (**Listing 19**).

Listing 19. The UFO class types (ufos.cs).

```
namespace CSharpOOPExample
{
  public class UFO
  {
    public void AbductHuman()
    { Console.WriteLine("Come here Earthling...");
  }

  public class MotherShip : UFO
  {
    public void AbductOtherUFOs()
    { Console.WriteLine
       ("You have violated the prime directive."); }
  }
}
```

At this point, we have two distinct hierarchies (car-types and UFO-types). When you look at the classes defined in each category, it might strike you that UFO, MotherShip and JamesBondCar could share common behavior in that they all have the ability to become airborne. If you wish to build a further association between these types, you won't get very far with classical polymorphism given that virtual and abstract methods in an ABC are only useful if the types are in the same hierarchy (which is not the case here). However, if you pull the common functionality into an interface definition, you can selectively attach this behavior on a type-by-type basis. To illustrate, define an interface named IAirVehicle within a file named interfaces.cs (**Listing 20**).

Listing 20. The IAirVehicle interface (interfaces.cs)

```
namespace CSharpOOPExample
{
  public interface IAirVehicle
  {
    void Hover();
    bool CanLeaveAtmosphere {get;}
  }
}
```

Here, IAirVehicle defines a single method and a read only property. Again given that interfaces are nothing but a named collection of abstract methods, we have no implementation details, no access modifier (interface methods are always public) and no member variables.

Once an interface is defined, it may now be implemented by each type that should support this behavior. First, the UFOs (**Listing 21**).

Listing 21. Implementing IAirVehicle on the UFO types.

```
namespace CSharpOOPExample
{
  public class UFO : IAirVehicle
  {
...
  // Mark IAirVehicles.Hover() as virtual
  // to allow derived types to modify this
  // behavior.
    public virtual void Hover()
    { Console.WriteLine
      ("Hovering and observing humans..."); }
    public bool CanLeaveAtmosphere {get {return true;} }
  }

// Because MotherShip derives from UFO,
// it is automatically IAirVehicle
// compatible.
  public class MotherShip : UFO
  {
...
    public override void Hover()
    { Console.WriteLine
      ("Hovering and observing other UFOs..."); }
  }
}
```

Implementing an interface is an all or nothing proposition. Given that UFO states that it supports IAirVehicle, it is now obligated to implement Hover() and CanLeaveAtmosphere. Notice that when UFO does implement the Hover() method, it marks this member as *virtual*. Thus, the derived MotherShip is free to redefine how it will implement this functionality while still being type compatible with IAirVehicle. Now, let's implement this same interface on JamesBondCar (**Listing 22**).

Listing 22. Implementing IAirVehicle on JamesBondCar.

```
public class JamesBondCar : SportsCar, IAirVehicles
{
...
  public void Hover()
  { Console.WriteLine
    ("Observing GoldFinger and OddJob..."); }
  public bool CanLeaveAtmosphere {get {return false;} }
}
```

Note that when you wish to explicitly mark a type's base class as well as implement some set of interfaces, you simply make use of a comma-delimited list (the first item after the semi-colon used on the class definition will always mark the base class). Given that UFO derives directly from System.Object, we can simply list the set of supported interfaces without explicitly deriving from Object (as this is assumed).

Interfaces in Action

At this point, we have three different classes (in distinct hierarchies), which implement the same interface. Now then, what is the benefit of doing so? First of all, you can declare an array of types that implement a given interface to exercise interface-based polymorphism (**Listing 23**).

Listing 23. Interface types as arrays.

```
// Create an array of IAirVehicle compatible types.
IAirVehicle[] myFlyingObjects = new IAirVehicle[4];
myFlyingObjects[0] =
  new JamesBondCar("Bimmer", 120, "*Classified*");
myFlyingObjects[1] = new UFO();
myFlyingObjects[2] = new UFO();
myFlyingObjects[3] = new MotherShip();

foreach(IAirVehicle aVCompatibleType in myFlyingObjects)
{
  Console.WriteLine("Can leave atmosphere? {0}",
    aVCompatibleType.CanLeaveAtmosphere);
}
```

To deepen your appreciation of interface programming techniques, assume we now which to create a System.Collections.ArrayList type within our Main() method. Because the Add() method of the ArrayList type is prototyped to take System.Objects, you can add literally anything into the container (**Listing 24**).

Listing 24. Populating our ArrayList with numerous things...

```
ArrayList allMyStuff = new ArrayList();
allMyStuff.Add(new JamesBondCar());
allMyStuff.Add(22);
allMyStuff.Add("Go Baldy, it's your birthday...");
allMyStuff.Add(new MiniVan());
allMyStuff.Add(new MotherShip());
allMyStuff.Add(false);

foreach(object o in allMyStuff)
{
  if(o is IAirVehicle)
    ((IAirVehicle)o).Hover();
  else
    Console.WriteLine("sorry, not an air vehicle.");
}
```

Here, the allMyStuff object contains numerous unrelated items (JamesBondCars, System.Int32s, System.String types, System.Booleans and so on), however we are able to investigate each sub-item using the 'is' (or 'as') keyword to dynamically discover which items are IAirVehicle compatable. If the current item is IAirVehicle aware, we cast the object accordingly and call the Hover() method. Again, the beauty of the interface is the fact that we can inject polymorphism across diverse class hierarchies. Furthermore, given the language agnostic nature of .NET, it is commonplace to define an interface in one programming language and implement it within another.

Like garbage collection, there are numerous other topics regarding interface based programming techniques under the .NET platform (explicit interface implementation, interface hierarchies, multiple inheritance of interface types and whatnot), however we'll need to call it a day for the time being.

Wrap Up

In this installment of *Casting Your .NET* you examined how the C# programming language contends with the pillars of OOP. Once you drilled through the basics of class definition and object allocation, you saw how to define type properties in the syntax of C#. Next you created a hierarchy of types using classical inheritance, and injected some polymorphic behavior using abstract methods and interface implementation. In the next article, I'll complete this C# primer by examining the remaining .NET types (enumerations, structures and delegates). See ya next issue, and happy hacking.

By Tom Djajadiningrat

Developing Electronic Hardware on Mac OS X

Schematic Capture and PCB Routing

Introduction

Contrary to popular belief amongst engineers using machines from the evil empire, it is quite well possible to develop electronics hardware on the Mac. In true Mac fashion, there are much fewer solutions, but those that are there are pretty good quality. What follows is a list of pointers to Mac software for schematic capture (the process of drawing electronic hardware diagrams, see **Figure 1**) and PCB routing (the process of laying out components and tracks on printed circuit boards, see **Figure 2**).

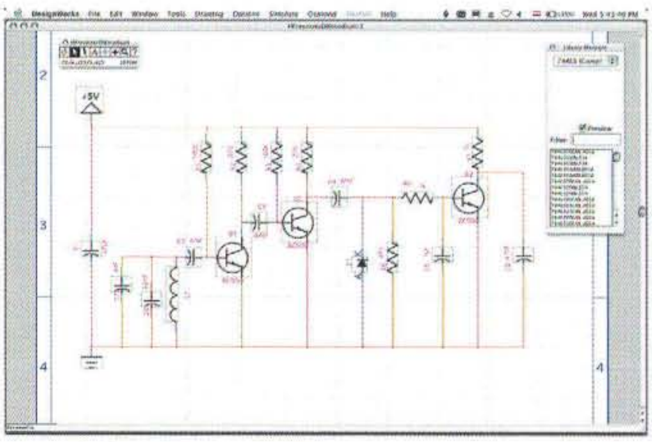


Figure 1: *Schematic capture*

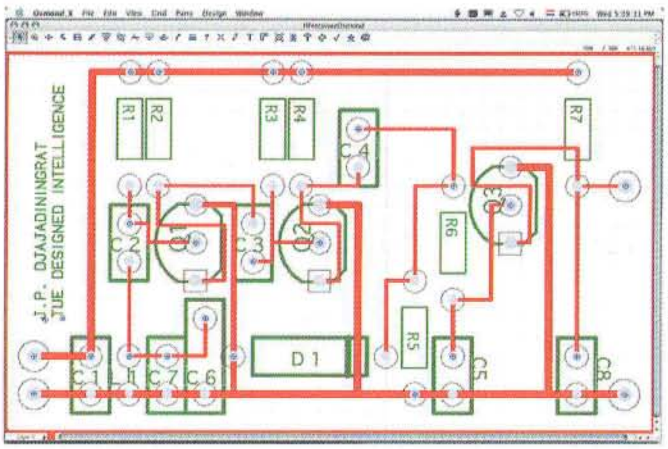


Figure 2: *Printed circuit board layout*

Capilano Computing

http://www.capilano.com

This company offers three packages: DesignWorks, DesignWorks Lite and Logic Works.

DesignWorks

(From 395 USD; save disabled evaluation version available)

This is Capilano's flagship product. It is purely a schematic capture and simulation program and cannot be used for laying out PCBs. However, it can export parts lists (specifying the components) and net lists (specifying the connections between components) to a great many PCB layout packages. These include not only software which runs on the Mac such as McCAD and Douglas, but also 'evil empire' only packages such as OrCAD and PADS.

Tom Djajadiningrat is a member of the Designed Intelligence Group, a research and teaching facility of Eindhoven University of Technology, which focuses on the design of intelligent products and services. You can reach him at j.p.djajadiningrat@tue.nl

DesignWorks Lite
(39.95 USD; 30 days evaluation version is free)

DesignWorks Lite does only schematic capture. It does not do simulation. More importantly, however, the parts/net list export functionality is missing. This makes DesignWorks Lite kind of useless for developing printed circuit boards: the essential link between schematic capture and PCB layout is absent. Of course, if you need to merely document electronic circuitry or have your circuit boards laid out by someone else then it may suit you just fine.

LogicWorks
(approx. 80 USD)

Luckily, Capilano offer something that sits in between DesignWorks and DesignWorks Lite: LogicWorks. Targeted originally at the educational market, it is bundled with a book and available from Amazon. Like DesignWorks, LogicWorks offers schematic capture and simulation, though with far fewer parts libraries. Unlike DesignWorks Lite, LogicWorks does offer export of parts and net lists, but only in a single format. You have to massage the parts and net lists into a format your PCB software understands, either using search and replace with a text editor (say BBEdit) or by parsing the text using your favourite programming language. Another caveat is that LogicWorks 4.0 is currently Mac OS9 only though rumours have it that version 4.5 will be OSX native and is soon to appear. If a Mac OS9 version is ok, you may want to hunt about in the secondhand section of Amazon.

Douglas

- http://www.douglas.com/
- Douglas Professional System 25 USD
- Professional Plus System 495 USD

Douglas are not only a software company but also have manufacturing facilities for printed circuit boards which puts them in an interesting position. They offer two versions of their software. One is called Douglas Professional Plus. This program is for PCB routing which does autorouting (automatically laying out the tracks). Its natural companion seems to be Capilano's DesignWorks which can even be bought through Douglas. Douglas Professional Plus exports layouts in Gerber format and therefore you can have your printed circuit boards made at any company you like.

Douglas also make a second version of their software which is simply called Douglas Professional. It does the same as the Professional Plus system but it cannot export to Gerber. This means that you need to use Douglas' manufacturing facilities. To this end the software contains a job order and price quote module.

Whether it is economical to go for the Douglas Professional system and get your prints made at Douglas of course depends on a price comparison between Douglas and your local PCB manufacturer but also on where you are based (postage) and how frequently you need prints and in what volumes.

McCad

- http://www.mccad.com/
- EDS-1 (schematic capture, PCB layout, Gerber export) $1495
- EDS-2 (EDS-1 + simulation) $1895
- TK-1 (autorouter) $895
- EDS-Lite (free version of EDS-1 with a 200pin limit and smaller board size)

McCad offer packages for schematic capture, simulation, PCB layout, autorouting and Gerber export, either separately or combined. McCad seem to be the only ones who offer all these components, developed in-house for the Mac platform. You have to cough up some serious money to get this kind of integrated workflow though. If you can live with a 200pin limit you can actually get EDS-Lite for free.

Osmond

http://www.swcp.com/~jchavez/osmond.html
(free download)

Osmond is a currently free PCB layout tool which is still in beta. Though this may sound unattractive, Osmond appears to be much more stable than many so-called final releases. In fact, judging from the version history Osmond seems to have been in beta for more than five years. Personally, what I like about it, is that it does not suffer from featuritis. It seems to do what it is supposed to do but no more. The interface feels very clean and intuitive and the tutorial was enough to get me started in about two hours. Should you get stuck there is always the Osmond Yahoo! Group: http://groups.yahoo.com/group/osmondpcb/. Osmond exports Gerber/Excellon files so you can have your boards made anywhere you like. One of Osmond's drawbacks is that it does not do autorouting. Though it can do rat's nests and can indicate which pads need to be connected, you have to route the tracks around the board yourself as it cannot do it automatically. As long as the boards are not too complex, this need not be a problem. An example of a small print designed in Osmond can be seen in **Figure 2-5**. Osmond comes with various parts libraries. What I found somewhat tedious was making part names in LogicWorks' parts/net lists match the part names in Osmond's libraries.

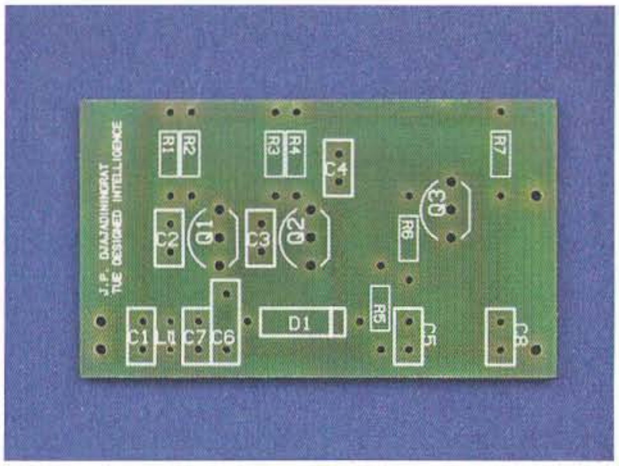


Figure 3: *Silkscreen side of the one-sided print resulting from the Osmond file shown in* ***Figure 2***

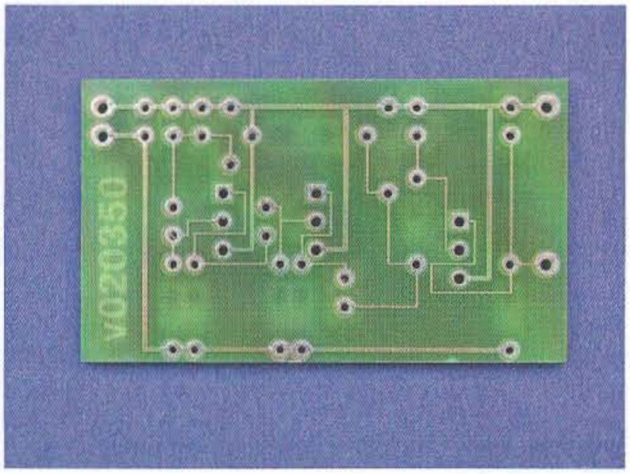


Figure 4: *The side with the tracks.*

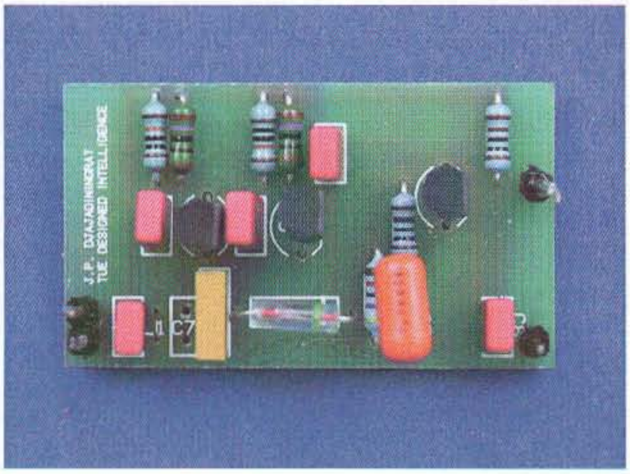


Figure 5: *The completed print with components*

Electric

(open source; pre-built OSX binary: 40 USD)
http://www.staticfreesoft.com/

Electric is somewhat different from the other packages mentioned here as it was not originally developed for the Macintosh. It is also the only package which I haven't tried for myself. Electric is open source, cross-platform software for schematic capture, simulation and PCB layout. However, compiling the source using XCode or CodeWarrior is apparently non-trivial: one has to do quite a bit of tweaking. The recommended compiler is Qt by Trolltech (www.trolltech.com). Although Qt is far from free (1550USD), a 30 day evaluation version is available. If you are not into compiling Electric yourself, you can always order a CD with a pre-built OSX binary.

Gerber Viewers

To send your design to a print manufacturer it needs to be in a file format called Gerber. It is always a nice sanity check to be able to view these files before you send them off. There are two Gerber viewers available. One is GerberReader which is a utility included with Douglas Professional. You may find that this utility alone makes it worth to spend $25 on Douglas Professional. The other is Gerber2PDF, a Python command-line utility which you can get from the Osmond home page. It takes a little effort to install but it does actually work very well, with different board layers being output to separate pages of a multipage PDF document. When you open such a document with Apple's Preview app, you can simply switch layers by pressing page icons in the drawer.

Summary

So there you have it, there is quite some choice actually. Some of these packages have been around since the very start of the Mac: obviously the Macintosh GUI worked miracles for such spatial tasks as schematic capture and PCB routing. What this list of pointers does not address is the user experience: each of these packages has its own idiosyncrasies. Therefore I would strongly recommend that you try out a few of these packages for yourself before you jump. The nice thing is that virtually all of them are available in evaluation versions of their software so that you can get a feel for the full package whilst not spending more than 25-50 US Dollars.

by Neil Ticktin, Publisher, MacTech Magazine

SoundSticks II by harman multimedia

As you might expect, I'm definitely a computer guy ... 6 computers in one house with 4 people. Lights, sprinklers, and more are controlled by one of the machines, and of course, we've now added a G5 for a "media station" since we've been averaging more than 1250 pictures per month, since 2001, with my trusty Nikon D100.

All this technology aside, taking pictures is not about being a geek ... it's about capturing an emotion. And, while stills capture the moment, stringing them together and putting them to music is one of the most rewarding experiences I've ever had. There's nothing like putting a teacher and classroom full of parents in tears ... or hearing a six year old yell out the names of their teammates when they see them on the screen.

In my set up, my G5 sits several feet to the side of its screen and keyboard, and while the G5 is a great machine, listening to music out of it while composing a slide show or video ... let alone just listening to iTunes for pleasure ... just doesn't cut it.

I started to look around at options just to get the speakers up next to the screen where I'm working. I quickly realized that while technology has come a long way, cheap speakers for your computer still sound cheap ... they sound like computer speakers. While I don't get the time to listen to music as I once did, I still do have a discerning ear and "computer speakers" were not going to cut it.

I looked a variety of the options out there, including harman multimedia's assortment. I always liked the SoundSticks design and layout (especially the small form factor on my desk), but I didn't like the need to go through USB and software to handle my speaker output. I'm an "old school" guy ... we have audio outputs, let's use them.

Enter SoundSticks II

harman multimedia has taken a great product in SoundSticks, and made it even better with SoundSticks II by taking to a standard analog audio connectivity and touch volume control.

As any audio aficionado will tell you, sound is a personal thing. It's best to listen to speakers side by side using the exact same source to determine what is attractive to you. When buying speakers for your computer, this is relatively difficult. That said, I had the opportunity to listen to a variety of harman and JBL speakers (as well as other brands), and to my taste, the SoundSticks II were hands-down the winner.

To me, SoundSticks II were well worth a bit of extra money (retail for $199) for the clarity, accuracy, and fullness of the sound. If you want to just go with speakers without having the opportunity to do a side by side sample as I have, these are winners.

The details...

SoundSticks II have three main parts -- two satellite speakers that are tall and narrow, and one larger subwoofer, which optimally sits below your desk. Frankly, the hardest thing about setting them up is unpacking the box. :)

The only issue that I had in set up is that it took me a couple of minutes to find the volume controls the documentation mentioned (at the base of one of the satellites). It would have been nice to see this better in the documentation.

These speakers can pack a wallop ... enough of one that when i first set it up, my kids on the other side of the house made me turn them down. :(

Conclusion

You've spent a good chunk of money on your system ... if you are either listening to iTunes or making slide shows or movies, you have to get the full effect by upgrading your sound. SoundSticks II is, hands-down, the best solution that I found at a moderate price.

ADVERTISER/ PRODUCT LIST

List of Advertisers

List of Products

Multiple formats. Multiple platforms. Complex installers.

Aladdin solves the compression and installation puzzle.

Trying to figure out how to handle multiple compression formats and platforms?
The StuffIt Engine solves the compression puzzle.

Aladdin's StuffIt Engine SDK:

- Adds value to your application by integrating powerful compression and encryption.
- Is the only tool that supports the StuffIt file format.
- Provides a single API that supports over 20 compression and encoding formats common on Macintosh, Windows, and Unix.
- Makes self-extracting archives for either Macintosh or Windows.
- Available for Macintosh, Windows, Linux, or Solaris.

Licenses start as low as $99/year

To learn more, visit: www.stuffit.com/sdk/

StuffIt Engine SDK™
The power of StuffIt in your software.

Looking for the easiest and fastest way to build an installer?
StuffIt InstallerMaker completes your puzzle.

It's not enough just to write solid code anymore. You still have to write an installer for your users. StuffIt InstallerMaker makes it simple and effective.

- StuffIt InstallerMaker gives you all the tools you need to install, uninstall, resource-compress or update your software in one complete, easy-to-use package.
- Add marketing muscle to your installers by customizing your electronic registration form to include surveys and special offers.
- Make demoware in minutes. Create Macintosh OS X and Macintosh Classic compatible installers with StuffIt InstallerMaker.

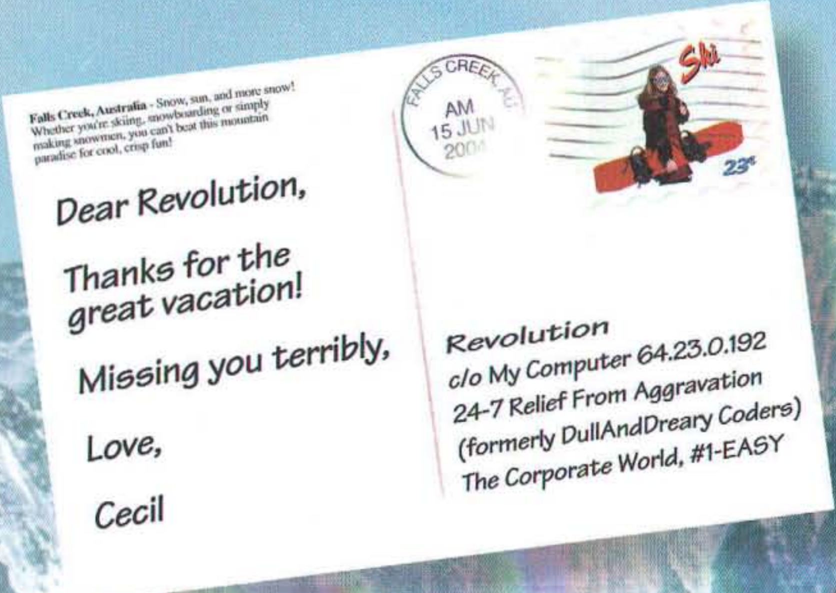
Prices start at $250

To learn more, visit: www.stuffit.com/installermaker/

StuffIt InstallerMaker™
The complete installation solution.™

Aladdin Systems

www.stuffit.com
(831) 761-6200


Falls Creek, Australia - Snow, sun, and more snow! Whether you're skiing, snowboarding or simply making snowmen, you can't beat this mountain paradise for cool, crisp fun!

FALLS CREEK, AU
AM
15 JUN
2004

Ski
23¢

Dear Revolution,

Thanks for the great vacation!

Missing you terribly,

Love,

Cecil

Revolution
c/o My Computer 64.23.0.192
24-7 Relief From Aggravation
(formerly DullAndDreary Coders)
The Corporate World, #1-EASY

Revolution: the English like language designed around the way you think.

Develop fast and deliver at the click of a button on 12 platforms, including Mac OS X, Windows and Linux. Support for XML, SQL databases, video capture, Unicode, Reports, faceless CGI applications, and more.

For as little as $149 *you* can create applications, storyboard projects, manage your databases, build courseware, work with the Internet, create "quick and dirty" solutions to those niggling office tasks in minutes, ship polished programs in hours or days instead of weeks and months...

Thousands of developers have already joined the Revolution. Can you afford to wait?

Scale the peaks with Revolution!

RUNTIME REVOLUTION
User-Centric Development

Runtime Revolution • 91 Hanover Street • Edinburgh EH2 1DJ • UK
Phone +44 (0)131 718 4333 • Fax +44 (0) 845 458 8487 • www.runrev.com • Email info@runrev.com